

30th Annual European Symposium on Algorithms

ESA 2022, September 5–9, 2022, Berlin/Potsdam, Germany

Edited by

Shiri Chechik

Gonzalo Navarro

Eva Rotenberg

Grzegorz Herman



LIPICS

Editors

Shiri Chechik

Tel Aviv University, Israel
shiri.chechik@gmail.com

Gonzalo Navarro 

University of Chile, Santiago, Chile
gnavarro@dcc.uchile.cl

Eva Rotenberg 

Technical University of Denmark, Lyngby, Denmark
erot@dtu.dk

Grzegorz Herman 

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

ACM Classification 2012

Applied computing → Transportation; Computing methodologies → Linear algebra algorithms; Computing methodologies → Shared memory algorithms; Human-centered computing → Graph drawings; Information systems → Clustering; Information systems → Data structures; Mathematics of computing → Algebraic topology; Mathematics of computing → Approximation algorithms; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Combinatorics; Mathematics of computing → Discrete optimization; Mathematics of computing → Graph algorithms; Mathematics of computing → Graph coloring; Mathematics of computing → Graph theory; Mathematics of computing → Network flows; 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 → Trees; Networks → Network algorithms; Theory of computation → Algorithm design techniques; Theory of Computation-Analysis of algorithms and problem complexity; Theory of computation → Approximation algorithms analysis; Theory of computation → Bloom filters and hashing; Theory of computation → Branch-and-bound; Theory of computation → Communication complexity; Theory of computation → Complexity theory and logic; 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 → Exact and approximate computation of equilibria; Theory of computation → Facility location and clustering; Theory of computation → Fixed parameter tractability; Theory of computation → Formal languages and automata theory; Theory of computation → Graph algorithms analysis; Theory of computation → Integer Programming; Theory of computation → K-server algorithms; Theory of computation → Linear programming; Theory of computation → Network flows; Theory of computation → Online algorithms; Theory of computation → Packing and covering problems; Theory of computation → Parallel algorithms; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Pattern matching; Theory of computation → Problems, reductions and completeness; Theory of computation → Quantum computation theory; Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Random projections and metric embeddings; Theory of computation → Rounding techniques; Theory of computation → Scheduling algorithms; Theory of computation → Shared memory algorithms; Theory of computation → Shortest paths; Theory of computation → Sketching and sampling; Theory of computation → Solution concepts in game theory; Theory of computation → Sparsification and spanners; Theory of computation → Stochastic approximation; Theory of computation → Stochastic control and optimization; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Submodular optimization and polymatroids; Theory of computation → W hierarchy

ISBN 978-3-95977-247-1

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/978-3-95977-247-1>.

Publication date

September, 2022

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.2022.0

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>Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman</i>	0:xiii
Program Committees	
.....	0:xv–0:xvi
List of External Reviewers	
.....	0:xvii–0:xxi

Regular Papers

Enumerating Minimal Connected Dominating Sets	
<i>Faisal N. Abu-Khzam, Henning Fernau, Benjamin Gras, Mathieu Liedloff, and Kevin Mann</i>	1:1–1:15
Non-Adaptive Edge Counting and Sampling via Bipartite Independent Set Queries	
<i>Raghavendra Addanki, Andrew McGregor, and Cameron Musco</i>	2:1–2:16
Hardness of Token Swapping on Trees	
<i>Oswin Aichholzer, Erik D. Demaine, Matias Korman, Anna Lubiw, Jayson Lynch, Zuzana Masárová, Mikhail Rudoy, Virginia Vassilevska Williams, and Nicole Wein</i>	3:1–3:15
Tight Bounds for Online Matching in Bounded-Degree Graphs with Vertex Capacities	
<i>Susanne Albers and Sebastian Schubert</i>	4:1–4:16
TSP in a Simple Polygon	
<i>Henk Alkema, Mark de Berg, Morteza Monemizadeh, and Leonidas Theocharous</i> ..	5:1–5:14
Classical and Quantum Algorithms for Variants of Subset-Sum via Dynamic Programming	
<i>Jonathan Allcock, Yassine Hamoudi, Antoine Joux, Felix Klingelhöfer, and Miklos Santha</i>	6:1–6:18
Techniques for Generalized Colorful k -Center Problems	
<i>Georg Anegg, Laura Vargas Koch, and Rico Zenklusen</i>	7:1–7:14
Simple Streaming Algorithms for Edge Coloring	
<i>Mohammad Ansari, Mohammad Saneian, and Hamid Zarrabi-Zadeh</i>	8:1–8:4
Computing Smallest Convex Intersecting Polygons	
<i>Antonios Antoniadis, Mark de Berg, Sándor Kisfaludi-Bak, and Antonis Skarlatos</i> ..	9:1–9:13
The Price of Hierarchical Clustering	
<i>Anna Arutyunova and Heiko Röglin</i>	10:1–10:14
Bounding and Computing Obstacle Numbers of Graphs	
<i>Martin Balko, Steven Chaplick, Robert Ganian, Siddharth Gupta, Michael Hoffmann, Pavel Valtr, and Alexander Wolff</i>	11:1–11:13

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman

Leibniz International Proceedings in Informatics



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

Computing NP-Hard Repetitiveness Measures via MAX-SAT <i>Hideo Bannai, Keisuke Goto, Masakazu Ishihata, Shunsuke Kanda, Dominik Köppl, and Takaaki Nishimoto</i>	12:1–12:16
Online Metric Allocation and Time-Varying Regularization <i>Nikhil Bansal and Christian Coester</i>	13:1–13:13
An Upper Bound on the Number of Extreme Shortest Paths in Arbitrary Dimensions <i>Florian Barth, Stefan Funke, and Claudius Proissl</i>	14:1–14:12
Galactic Token Sliding <i>Valentin Bartier, Nicolas Bousquet, and Amer E. Mouawad</i>	15:1–15:14
When Are Cache-Oblivious Algorithms Cache Adaptive? A Case Study of Matrix Multiplication and Sorting <i>Arghya Bhattacharya, Abiyaz Chowdhury, Helen Xu, Rathish Das, Rezaul A. Chowdhury, Rob Johnson, Rishab Nithyanand, and Michael A. Bender</i>	16:1–16:17
Simple Dynamic Spanners with Near-Optimal Recourse Against an Adaptive Adversary <i>Sayan Bhattacharya, Thatchaphol Saranurak, and Pattara Sukprasert</i>	17:1–17:19
Online Spanners in Metric Spaces <i>Sujoy Bhore, Arnold Filtser, Hadi Khodabandeh, and Csaba D. Tóth</i>	18:1–18:20
Sparse Temporal Spanners with Low Stretch <i>Davide Bilò, Gianlorenzo D’Angelo, Luciano Gualà, Stefano Leucci, and Mirko Rossi</i>	19:1–19:16
Resource Sharing Revisited: Local Weak Duality and Optimal Convergence <i>Daniel Blankenburg</i>	20:1–20:14
On the External Validity of Average-Case Analyses of Graph Algorithms <i>Thomas Bläsius and Philipp Fischbeck</i>	21:1–21:14
On Polynomial Kernels for Traveling Salesperson Problem and Its Generalizations <i>Václav Blažej, Pratibha Choudhary, Dušan Knop, Šimon Schierreich, Ondřej Suchý, and Tomáš Valla</i>	22:1–22:16
Maximizing Sums of Non-Monotone Submodular and Linear Functions: Understanding the Unconstrained Case <i>Kobi Bodek and Moran Feldman</i>	23:1–23:17
List Colouring Trees in Logarithmic Space <i>Hans L. Bodlaender, Carla Groenland, and Hugo Jacob</i>	24:1–24:15
Dynamic Coloring of Unit Interval Graphs with Limited Recourse Budget <i>Bartłomiej Bosek and Anna Zych-Pawlewicz</i>	25:1–25:14
Polynomial Kernel for Immersion Hitting in Tournaments <i>Łukasz Bożyk and Michał Pilipczuk</i>	26:1–26:17
A Systematic Study of Isomorphism Invariants of Finite Groups via the Weisfeiler-Leman Dimension <i>Jendrik Brachter and Pascal Schweitzer</i>	27:1–27:14

Faster Approximate Covering of Subcurves Under the Fréchet Distance <i>Frederik Brünig, Jacobus Conradi, and Anne Driemel</i>	28:1–28:16
Efficient Fréchet Distance Queries for Segments <i>Maïke Buchin, Ivor van der Hoog, Tim Ophelders, Lena Schlipf, Rodrigo I. Silveira, and Frank Staals</i>	29:1–29:14
Search-Space Reduction via Essential Vertices <i>Benjamin Merlin Bumpus, Bart M. P. Jansen, and Jari J. H. de Kroon</i>	30:1–30:15
Width Helps and Hinders Splitting Flows <i>Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams</i>	31:1–31:14
Counting Simplices in Hypergraph Streams <i>Amit Chakrabarti and Themistoklis Haris</i>	32:1–32:19
Approximation Algorithms for Continuous Clustering and Facility Location Problems <i>Deeparnab Chakrabarty, Maryam Negahbani, and Ankita Sarkar</i>	33:1–33:15
Distinct Elements in Streams: An Algorithm for the (Text) Book <i>Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel</i>	34:1–34:6
Approximate Circular Pattern Matching <i>Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba</i>	35:1–35:19
Multi-Dimensional Stable Roommates in 2-Dimensional Euclidean Space <i>Jiehua Chen and Sanjukta Roy</i>	36:1–36:16
Spanner Approximations in Practice <i>Markus Chimani and Finn Stutzenstein</i>	37:1–37:15
Determinants from Homomorphisms <i>Radu Curticapean</i>	38:1–38:7
Conditional Lower Bounds for Dynamic Geometric Measure Problems <i>Justin Dallant and John Iacono</i>	39:1–39:17
A Simpler QPTAS for Scheduling Jobs with Precedence Constraints <i>Syamantak Das and Andreas Wiese</i>	40:1–40:11
A Polynomial-Time Algorithm for 1/3-Approximate Nash Equilibria in Bimatrix Games <i>Argyrios Deligkas, Michail Fasoulakis, and Evangelos Markakis</i>	41:1–41:14
Near Optimal Algorithm for Fault Tolerant Distance Oracle and Single Source Replacement Path Problem <i>Dipan Dey and Manoj Gupta</i>	42:1–42:18
Fast Computation of Zigzag Persistence <i>Tamal K. Dey and Tao Hou</i>	43:1–43:15
Turbocharging Heuristics for Weak Coloring Numbers <i>Alexander Dobler, Manuel Sorge, and Anaïs Villedieu</i>	44:1–44:18

A Local Search Algorithm for Large Maximum Weight Independent Set Problems <i>Yuanyuan Dong, Andrew V. Goldberg, Alexander Noe, Nikos Parotsidis, Mauricio G.C. Resende, and Quico Spaen</i>	45:1–45:16
SAT Backdoors: Depth Beats Size <i>Jan Dreier, Sebastian Ordyniak, and Stefan Szeider</i>	46:1–46:18
Finding a Cluster in Incomplete Data <i>Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider</i>	47:1–47:14
Lyndon Arrays Simplified <i>Jonas Ellert</i>	48:1–48:14
Learning-Augmented Query Policies for Minimum Spanning Tree with Uncertainty <i>Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlöter</i>	49:1–49:18
Faster Exponential-Time Approximation Algorithms Using Approximate Monotone Local Search <i>Bariş Can Esmer, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma</i>	50:1–50:19
Intersection Searching Amid Tetrahedra in 4-Space and Efficient Continuous Collision Detection <i>Esther Ezra and Micha Sharir</i>	51:1–51:17
Submodular Maximization Subject to Matroid Intersection on the Fly <i>Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen</i>	52:1–52:14
There and Back Again: On Applying Data Reduction Rules by Undoing Others <i>Aleksander Figiel, Vincent Froese, André Nichterlein, and Rolf Niedermeier</i>	53:1–53:15
Improved Search of Relevant Points for Nearest-Neighbor Classification <i>Alejandro Flores-Velazco</i>	54:1–54:10
Longest Cycle Above Erdős–Gallai Bound <i>Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov</i>	55:1–55:15
Improved Polynomial-Time Approximations for Clustering with Minimum Sum of Radii or Diameters <i>Zachary Friggstad and Mahya Jamshidian</i>	56:1–56:14
Simple Worst-Case Optimal Adaptive Prefix-Free Coding <i>Travis Gagie</i>	57:1–57:5
Taming Graphs with No Large Creatures and Skinny Ladders <i>Jakub Gajarský, Lars Jaffke, Paloma T. Lima, Jana Novotná, Marcin Pilipczuk, Paweł Rzażewski, and Uéverton S. Souza</i>	58:1–58:8
Faster Path Queries in Colored Trees via Sparse Matrix Multiplication and Min-Plus Product <i>Younan Gao and Meng He</i>	59:1–59:15
Computing the 4-Edge-Connected Components of a Graph: An Experimental Study <i>Loukas Georgiadis, Giuseppe F. Italiano, and Evangelos Kosinas</i>	60:1–60:16
Algorithmic Meta-Theorems for Combinatorial Reconfiguration Revisited <i>Tatsuya Gima, Takehiro Ito, Yasuaki Kobayashi, and Yota Otachi</i>	61:1–61:15

Efficient Recognition of Subgraphs of Planar Cubic Bridgeless Graphs <i>Miriam Goetze, Paul Jungeblut, and Torsten Ueckerdt</i>	62:1–62:14
Adaptive-Adversary-Robust Algorithms via Small Copy Tree Embeddings <i>Bernhard Haepler, D. Ellis Hershkowitz, and Goran Zuzic</i>	63:1–63:14
Hedonic Games and Treewidth Revisited <i>Tesshu Hanaka and Michael Lampis</i>	64:1–64:16
Fine-Grained Complexity Lower Bounds for Families of Dynamic Graphs <i>Monika Henzinger, Ami Paz, and A. R. Sricharan</i>	65:1–65:14
$O(1)$ Steiner Point Removal in Series-Parallel Graphs <i>D. Ellis Hershkowitz and Jason Li</i>	66:1–66:17
Chromatic k -Nearest Neighbor Queries <i>Thijs van der Horst, Maarten Löffler, and Frank Staals</i>	67:1–67:14
Maximum Weight b -Matchings in Random-Order Streams <i>Chien-Chung Huang and François Sellier</i>	68:1–68:14
Embedding Phylogenetic Trees in Networks of Low Treewidth <i>Leo van Iersel, Mark Jones, and Mathias Weller</i>	69:1–69:14
Vertex Sparsifiers for Hyperedge Connectivity <i>Han Jiang, Shang-En Huang, Thatchaphol Saranurak, and Tian Zhang</i>	70:1–70:13
Approximation Algorithms for Round-UFP and Round-SAP <i>Debajyoti Kar, Arindam Khan, and Andreas Wiese</i>	71:1–71:19
Optimizing Safe Flow Decompositions in DAGs <i>Shahbaz Khan and Alexandru I. Tomescu</i>	72:1–72:17
Scheduling Kernels via Configuration LP <i>Dušan Knop and Martin Koutecký</i>	73:1–73:15
Abstract Morphing Using the Hausdorff Distance and Voronoi Diagrams <i>Lex de Kogel, Marc van Kreveld, and Jordi L. Vermeulen</i>	74:1–74:16
Average Sensitivity of the Knapsack Problem <i>Soh Kumabe and Yuichi Yoshida</i>	75:1–75:14
Cardinality Estimation Using Gumbel Distribution <i>Aleksander Łukasiewicz and Przemysław Uznański</i>	76:1–76:13
(In-)Approximability Results for Interval, Resource Restricted, and Low Rank Scheduling <i>Marten Maack, Simon Pukrop, and Anna Rodriguez Rasmussen</i>	77:1–77:13
Localized Geometric Moves to Compute Hyperbolic Structures on Triangulated 3-Manifolds <i>Clément Maria and Owen Rouillé</i>	78:1–78:13
Computing Treedepth in Polynomial Space and Linear FPT Time <i>Wojciech Nadara, Michał Pilipeczuk, and Marcin Smulewicz</i>	79:1–79:14
The Pareto Cover Problem <i>Bento Natura, Meike Neuwöhner, and Stefan Weltge</i>	80:1–80:12

A Unified Framework for Hopsets <i>Ofer Neiman and Idan Shabat</i>	81:1–81:13
Data Structures for Node Connectivity Queries <i>Zeev Nutov</i>	82:1–82:12
Improved Bounds for Online Balanced Graph Re-Partitioning <i>Rajmohan Rajaraman and Omer Wasim</i>	83:1–83:15
An Empirical Evaluation of k -Means Coresets <i>Chris Schwegelshohn and Omar Ali Sheikh-Omar</i>	84:1–84:17
An Improved Algorithm for Finding the Shortest Synchronizing Words <i>Marek Szykula and Adam Zyzik</i>	85:1–85:15
Fast RSK Correspondence by Doubling Search <i>Alexander Tiskin</i>	86:1–86:10
Insertion Time of Random Walk Cuckoo Hashing below the Peeling Threshold <i>Stefan Walzer</i>	87:1–87:11
ParGeo: A Library for Parallel Computational Geometry <i>Yiqiu Wang, Rahul Yesantharao, Shangdi Yu, Laxman Dhulipala, Yan Gu, and Julian Shun</i>	88:1–88:19
Combining Predicted and Live Traffic with Time-Dependent A^* Potentials <i>Nils Werner and Tim Zeitz</i>	89:1–89:15
Approximating Dynamic Time Warping Distance Between Run-Length Encoded Strings <i>Zoe Xi and William Kuszmaul</i>	90:1–90:19
Correlated Stochastic Knapsack with a Submodular Objective <i>Sheng Yang, Samir Khuller, Sunav Choudhary, Subrata Mitra, and Kanak Mahadik</i>	91:1–91:14
Faster Algorithm for Unique $(k, 2)$ -CSP <i>Or Zamir</i>	92:1–92:13

■ Preface

This volume contains the extended abstracts selected for presentation at ESA 2022, the 30th European Symposium on Algorithms. The event was organized by the Hasso Plattner Institute, Potsdam, Germany, as a part of ALGO 2022, on September 5–7, 2022.

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. A new track, S for Simplicity, was added this year, inviting contributions that simplify algorithmic results. We find that simpler algorithms are easier to implement, bridging the gap between theory and practice, and we find that new simple or elegant proofs are easier to understand and to teach, and may contain interesting new insights whose relevance only the future will reveal.

In response to the call for papers for ESA 2022, 300 papers were submitted, 214 for Track A, 46 for Track B, and 40 for Track S. Paper selection was based on originality, technical quality, exposition quality, and relevance. Each paper received at least three reviews. The program committees selected 92 papers for inclusion in the program: 68 from Track A, 14 from Track B, and 10 for Track S, yielding an overall acceptance rate of about 30%. The presentations of the accepted papers, together with two invited talks by Virginia Vassilevska (MIT) and Simon Puglisi (U. Helsinki), promise to make up a very exciting program.

The European Association for Theoretical Computer Science (EATCS) sponsored best paper and best student paper awards. A submission was eligible for the best student paper award if all authors were doctoral, master, or bachelor students at the time of submission. For track A, the best paper award was given to Stefan Walzer for the paper Insertion Time of Random Walk Cuckoo Hashing below the Peeling Threshold, and the best student paper award to Zoe Xi and William Kuszmaul for the paper Approximating Dynamic Time Warping Distance Between Run-Length Encoded Strings. For track B, the best paper award was given to Chris Schwiegelshohn and Omar Ali Sheikh-Omar for the paper An Empirical Evaluation of k -Means Coresets, and the best student paper award to Tim Zeitz and Nils Werner for the paper Combining Predicted and Live Traffic with Time-Dependent A* Potentials. The best paper award for track S was given to Alejandro Flores-Velazco for the paper Improved Search of Relevant Points for Nearest-Neighbor Classification—this was also the best student paper for this track.

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.

Information on past ESA symposia, including locations and proceedings, is maintained at <http://esa-symposium.org>.



■ Program Committees

Track A (Design and Analysis) Program Committee

- Mikkel Abrahamsen, University of Copenhagen
- Peyman Afshani, Aarhus University
- Pankaj K. Agarwal, Duke University
- Sepehr Assadi, Rutgers University
- Per Austrin, KTH
- Leonid Barenboim, The Open University of Israel
- Surender Baswana, IIT Kanpur
- Maike Buchin, Ruhr University Bochum
- Jaroslaw Byrka, University of Wrocław
- Diptarka Chakraborty, National University of Singapore
- Shiri Chechik (chair), Tel Aviv University
- Vincent Cohen-Addad, Google Research
- Mark de Berg, TU Eindhoven
- Mahsa Derakhshan, UC Berkley and Northeastern University
- Michael Dinitz, Johns Hopkins University
- Michal Dory, ETH Zurich
- Matthias Englert, University of Warwick
- Thomas Erlebach, Durham University
- Fedor Fomin, University of Bergen
- Dimitris Fotakis, National Technical University of Athens
- Hsin Hao Su, Boston College
- Martin Hoefer, Goethe University
- Ravishankar Krishnaswamy, Microsoft Research
- Janardhan Kulkarni, Microsoft Research
- Divyarthi Mohan, Tel Aviv University
- Shay Mozes, Reichman University
- Wolfgang Mulzer, Freie Universität Berlin
- Ofer Neiman, Ben-Gurion University
- Aleksandar Nikolov, University of Toronto
- Sigal Oren, Ben-Gurion University
- Fahad Panolan, IIT Hyderabad
- Adi Rosén, FILOFOCS – CNRS
- Sushant Sachdeva, University of Toronto
- Stefan Schmid, University of Vienna and TU Berlin
- Roy Schwartz, Technion
- Bruce Shepherd, University of British Columbia
- Shay Solomon, Tel Aviv University
- Xiaorui Sun, University of Illinois
- Dimitrios Thilikos, LIRMM, Université de Montpellier, CNRS
- Ohad Trabelsi, The University of Michigan
- Oren Weimann, University of Haifa
- Philip Wellnitz, Max Planck Institute for Informatics
- Raphael Yuster, University of Haifa

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman

Leibniz International Proceedings in Informatics



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

Track B (Engineering and Applications) Program Committee

- Diego Arroyuelo, Universidad Técnica Federico Santa María
- Philip Bille, Danmarks Tekniske Universitet
- Thomas Bläsius, Karlsruhe Institute of Technology
- Christina Boucher, University of Florida
- Sándor Fekete, Technische Universität Braunschweig
- José Fuentes-Sepúlveda, Universidad de Concepción
- Gramoz Goranci, Universitat Wien
- Giuseppe Italiano, Università degli studi di Roma “Tor Vergata”
- Shweta Jain, University of Utah
- Dominik Kempa, Stony Brook University
- Veli Mäkinen, University of Helsinki
- Catherine McGeoch, Amherst College
- David Mount, University of Maryland
- Gonzalo Navarro (chair), Universidad de Chile
- Steven Skiena, Stony Brook University
- Matthias Stallmann, North Carolina State University

Track S (Simplicity) Program Committee

- Josh Alman, Columbia University
- Michael Bender, Stony Brook University
- Karl Bringmann, Saarland University
- Raphaël Clifford, University of Bristol
- Anne Driemel, Universität Bonn
- Paweł Gawrychowski, University of Wrocław
- Monika Henzinger, University of Vienna
- John Iacono, Université libre de Bruxelles
- Tomasz Kociumaka, University of California, Berkeley
- Irina Kostitsyna, Eindhoven University of Technology
- William Kuszmaul, Massachusetts Institute of Technology
- Rasmus Kyng, ETH Zürich
- Kitty Meeks, University of Glasgow
- Marcin Pilipczuk, University of Warsaw
- Kent Quanrud, Purdue University
- Eva Rotenberg (chair), Technical University of Denmark
- Shikha Singh, Williams College
- Jukka Suomela, Aalto University
- Haitao Wang, Utah State University
- Andreas Wiese, TU Munich
- Anna Zych-Pawlewicz, University of Warsaw

■ List of External Reviewers

Ahmed Abdelkader
Marek Adamczyk
Akanksha Agrawal
Saba Ahmadi
Jarno Alanko
Yeganeh Ali Mohammadi
Henk Alkema
Alexandr Andoni
Spyros Angelopoulos
Antonios Antoniadis
Tetsuya Araki
Emmanuel Arrighi
Martin Aumüller
Sayan Bandyopadhyay
Sandip Banerjee
Aritra Banik
Nikhil Bansal
Julien Baste
Aniket Basu Roy
Ulrich Bauer
Soheil Behnezhad
Benjamin Aram Berendsohn
Lorenzo Beretta
Giulia Bernardini
Aaron Bernstein
Nico Bertram
Ivona Bezakova
Amey Bhangale
Koustav Bhanja
Marcin Bienkowski
Greg Bodwin
Michaela Borzechowski
Nicolas Bousquet
Frederik Brüning
Kevin Buchin
Benjamin A. Burton
Martin Böhm
Edson Caceres
Linda Cai
Gruia Calinescu
Nairen Cao
Yixin Cao
Sankardeep Chakraborty
Parinya Chalermsook
Timothy M. Chan
Panagiotis Charalampopoulos
Krishnendu Chatterjee
Anamay Chaturvedi
Juhi Chaudhary
Prasad Chaugule
Li Chen
Yun Kuen Cheung
Riley Chinburg
Man Kwun Chiu
Aruni Choudhary
Keerti Choudhary
Amelia Chui
Eldon Chung
Jonas Cleve
Peter Clifford
Christian Coester
Avi Cohen
Jacobus Conradi
Guilherme D. Da Fonseca
Justin Dallant
Syamantak Das
Sami Davies
Mateus De Oliveira Oliveira
Argyrios Deligkas
Palash Dey
Martin Dietzfelbinger
Konstantinos Dogeas
Sally Dong
Valerio Dose
Jan Dreier
Ran Duan
Vida Dujmovic
Adrian Dumitrescu
Naveen Durvasula
Zdenek Dvorak
Talya Eden
Alon Efrat
Attila Egri-Nagy
Eduard Eiben
Mahmoud Elashmawi
Jonas Ellert
David Eppstein
Rolf Fagerberg

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman

Leibniz International Proceedings in Informatics



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

0:xviii List of External Reviewers

Alireza Farhadi
John Fearnley
Adrian Feilhauer
Andreas Emil Feldmann
Stefan Felsner
Hendrik Fichtenberger
Arnold Filtser
Omrit Filtser
Anja Fischer
Nick Fischer
Till Fluschnik
Marcelo Fonseca Faraj
Sebastian Forster
Florent Foucaud
Kyle Fox
Kshitij Gajjar
Karthik Gajulapalli
Arnab Ganguly
Yu Gao
Yuan Gao
Jugal Garg
Naveen Garg
Diego Gatica
Evangelia Gergatsouli
Mrinalkanti Ghosh
Prantar Ghosh
Sumanta Ghosh
Rohan Ghuge
Daniel Gibney
Yuval Gitlitz
Marc Glisse
Surbhi Goel
Marc Goerigk
Shay Golan
Siddharth Gollapudi
Sivakanth Gopi
Mayank Goswami
Roberto Grossi
Frank Gurski
Max Göttlicher
Inge Li Gørtz
Anselm Haak
Thekla Hamm
Sariel Har-Peled
David Harris
Ishay Haviv
Meng He
Stephan Held
Danny Hermelin
D. Ellis Hershkowitz
Karl Heuer
Lunjia Hu
Chien-Chung Huang
Shang-En Huang
Wenhan Huang
Sophie Huiberts
Thore Husfeldt
Tanmay Inamdar
Alexander Irribarra
Lars Jaffke
Wojciech Janczewski
Bart M. P. Jansen
Rajesh Jayaram
Shaofeng Jiang
Billy Jin
Ce Jin
Wenyu Jin
Adam Jozefiak
Paul Jungeblut
Kai Kahler
Alkis Kalavasis
John Kallaugher
Shahin Kamali
Frank Kammer
Lawqueen Kanesh
Mamadou Moustapha Kanté
Mong-Jen Kao
Haim Kaplan
Adam Karczmarz
Neel Karia
Andreas Karrenbauer
Karthik C. S.
Maximilian Katzmann
Alexander Kauer
Jenny Kaufmann
Jun Kawahara
Phillip Keldenich
Steven Kelk
Leon Kellerhals
Thomas Kesselheim
Balázs Keszegh
Shahbaz Khan
Kamyar Khodamoradi
Sándor Kisfaludi-Bak
Peter Kiss
Pieter Kleer

Linda Kleist	Jakub Łacki
Max Klimm	Aleksander Łukasiewicz
Nina Klobas	Ramanujan M. Sridharan
Katharina Klost	Cong Ma
Marina Knittel	Will Ma
Kristin Knorr	Calum MacRury
Tim Koglin	Sepideh Mahabadi
Sudeshna Kolay	Diptapriyo Majumdar
Hanna Komlos	Florin Manea
Matthias Konitzny	Pasin Manurangsi
Christian Konrad	Alberto Marchetti-Spaccamela
Tsvi Kopelowitz	Clément Maria
Tuukka Korhonen	Angeliki Mathioudaki
Dmitry Kosolobov	Simon Mauras
Lukasz Kowalik	Samuel McCauley
Laszlo Kozma	Andrew McGregor
Dominik Krupke	Moti Medina
Rajendra Kumar	Kurt Mehlhorn
Pascal Kunz	Arne Meier
O-Joung Kwon	Konstantina Mellou
Dominik Köppl	Arturo Merino
Bundit Laekhanukit	Laura Merker
Abhiruk Lahiri	Evi Micha
Benjamin Langmead	Till Miltzow
Thomas Lavastida	Pranabendu Misra
Ranko Lazic	James Mitchell
Hung Le	Antonio Molina Lovett
Euiwoong Lee	Morteza Monemizadeh
Mitchell Lee	Laure Morelle
Clément Legrand-Duchesne	Pat Morin
Pascal Lenzner	Amer Mouawad
Michael Lesnick	Anish Mukherjee
Michael Levet	Nabil Mustafa
Asaf Levin	Tobias Mömke
Huan Li	Torsten Mütze
Jason Li	Wojciech Nadara
Lawrence Li	Viswanath Nagarajan
Ray Li	Seffi Naor
Quanquan Liu	Guylain Naves
Yang Liu	Yasamin Nazari
William Lochet	Jesper Nederlof
Daniel Lokshtanov	Maryam Negahbani
Raul Lopes	Jelani Nelson
Changhong Lu	Prajakta Nimbhorkar
Xinhang Lu	Alexander Noe
Anna Lubiw	Ashkan Norouzi Fard
Xin Lyu	André Nusser
Maarten Löffler	Zeev Nutov

Johannes Obenaus
Eunjin Oh
Yoshio Okamoto
Karolina Okrasa
Hirotaka Ono
Yota Otachi
Joseph Paat
Rasmus Pagh
Katarzyna Paluch
Ioannis Panageas
Konstantinos Panagiotou
Orestis Papadigenopoulos
Evanthia Papadopoulou
Nikos Parotsidis
Boaz Patt-Shamir
Christophe Paul
Ami Paz
Pan Peng
Richard Peng
Anthony Perez
Michael Perk
Dominik Peters
Jannik Peters
Seth Pettie
Stephen Piddock
Théo Pierron
Karol Pokorski
Adam Polak
Tristan Pollner
Joseph Poremba
Nicola Prezza
Maximilian Probst
Evangelos Protopapas
Manish Purohit
Benjamin Raichel
Venkatesh Raman
Timothy Randolph
Felix Reidl
Malte Renken
Kilian Risse
Liam Roditty
Marcel Roeloffzen
Mohammad Roghani
Dennis Rohde
Lars Rohwedder
Jonathan Rollin
Will Rosenbaum
Marc Roth
Thomas Rothvoss
Ignaz Rutter
Emily Ryu
Andrew Ryzhikov
Paweł Rzażewski
Harald Räcke
Heiko Röglin
Arpan Sadhukhan
Danil Sagunov
Ashwin Sah
Toshiki Saitoh
Hamed Saleh
Thatchaphol Saranurak
Srinivasa Rao Satti
Ignasi Sau
Jonas Sauer
David Saulpic
Saket Saurabh
Dominik Scheder
Philipp Schepper
Kevin Schewior
Baruch Schieber
Marco Schmalhofer
Daniel Schmand
Arne Schmidt
Melanie Schmidt
Daniel Schoepflin
Masood Seddighin
Saeed Seddighin
Sandeep Sen
C. Seshadhri
Vihan Shah
Golnoosh Shahkarami
Mordechai Shalom
Roohani Sharma
Don Sheehy
Or Sheffet
Xiangkun Shen
Rodrigo Silveira
Sahil Singla
Makrand Sinha
Stavros Sintos
Stratis Skoulakis
Martin Skutella
Marek Sokółowski
Zhao Song
Jakob Spooner
Piyush Srivastava

Giannos Stamoulis
Aleksa Stankovic
Tatiana Starikovskaya
Alex Steiger
Sabine Storandt
Tord Stordalen
Ben Strasser
K. Subramani
Varun Suriyanarayana
Zoya Svitkina
Céline Swennenhuis
Seyed Ali Tabatabaee
Wai Ming Tai
Prafullkumar Tale
Jakub Tarnawski
Erin Taylor
Monique Teillaud
Jan Arne Telle
Sahas Thejaswi
Leonidas Theocharous
Clayton Thomas
Csaba Toth
Noam Touitou
Maria Tsalakou
Kostas Tsihlias
Abner Turkieltaub
Torsten Ueckerdt
Seeun William Umboh
Danny Vainstein
Ali Vakilian
Marc van Kreveld
Erik Jan van Leeuwen
André van Renssen
Shai Vardi
Nithin Varma
Giovanna Varricchio
Daniel Vaz
Santhoshini Velusamy
Laurent Viennot
Alexandre Vigny
Kai Vogelgesang
Hoa Vu
Nikhil Vyas
David Wajc
Bartosz Walczak
Chen Wang
Justin Ward
Philipp Warode
Kunihiro Wasa
Fan Wei
Nicole Wein
Omri Weinstein
Noah Weninger
Christopher Weyand
Sue Whitesides
Sebastian Wiederrecht
Marcus Wilhelm
Max Willert
Lucia Williams
Michal Włodarczyk
Piotr Wojciechowski
Sampson Wong
Xuan Wu
Christian Wulff-Nilsen
Karol Węgrzycki
Pucheng Xiong
Chao Xu
Helen Xu
Yinzhan Xu
Jie Xue
Jie Xue
Yutaro Yamaguchi
Katsuhisa Yamanaka
Alvin Yan Hong Yao
Mingquan Ye
Huacheng Yu
Weiqiang Yuan
Meirav Zehavi
Yibin Zhao
Yiming Zhao
Da Wei Zheng
Hongyu Zheng
Mingxian Zhong
Hang Zhou
Samson Zhou
Yuan Zhou



Enumerating Minimal Connected Dominating Sets

Faisal N. Abu-Khzam  

Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon

Henning Fernau   

Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Benjamin Gras  

Université d'Orléans, INSA Centre Val de Loire, LIFO EA 4022, France

Mathieu Liedloff  

Université d'Orléans, INSA Centre Val de Loire, LIFO EA 4022, France

Kevin Mann  

Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Abstract

The question to enumerate all (inclusion-wise) minimal connected dominating sets in a graph of order n in time significantly less than 2^n is an open question that was asked in many places. We answer this question affirmatively, by providing an enumeration algorithm that runs in time $\mathcal{O}(1.9896^n)$, using polynomial space only. The key to this result is the consideration of this enumeration problem on 2-degenerate graphs, which is proven to be possible in time $\mathcal{O}(1.9767^n)$. Apart from solving this old open question, we also show new lower bound results. More precisely, we construct a family of graphs of order n with $\Omega(1.4890^n)$ many minimal connected dominating sets, while previous examples achieved $\Omega(1.4422^n)$. Our example happens to yield 4-degenerate graphs. Additionally, we give lower bounds for the previously not considered classes of 2-degenerate and of 3-degenerate graphs, which are $\Omega(1.3195^n)$ and $\Omega(1.4723^n)$, respectively.

We also address essential questions concerning output-sensitive enumeration. Namely, we give reasons why our algorithm cannot be turned into an enumeration algorithm that guarantees polynomial delay without much efforts. More precisely, we prove that it is NP-complete to decide, given a graph G and a vertex set U , if there exists a minimal connected dominating set D with $U \subseteq D$, even if G is known to be 2-degenerate. Our reduction also shows that even any subexponential delay is not easy to achieve for enumerating minimal connected dominating sets. Another reduction shows that no FPT-algorithms can be expected for this extension problem concerning minimal connected dominating sets, parameterized by $|U|$. This also adds one more problem to the still rather few natural parameterized problems that are complete for the class W[3]. We also relate our enumeration problem to the famous open HITTING SET TRANSVERSAL problem, which can be phrased in our context as the question to enumerate all minimal dominating sets of a graph with polynomial delay by showing that a polynomial-delay enumeration algorithm for minimal connected dominating sets implies an affirmative algorithmic solution to the HITTING SET TRANSVERSAL problem.

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases enumeration problems, connected domination, degenerate graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.1

Related Version *Full Version*: <https://doi.org/10.48550/arXiv.2205.00086> [1]

Acknowledgements Henning Fernau likes to thank the Université d'Orléans for inviting him as a *professeur invité* in 2021; on these visits, much of the research was started.



© Faisal N. Abu-Khzam, Henning Fernau, Benjamin Gras, Mathieu Liedloff, and Kevin Mann; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 1; pp. 1:1–1:15

Leibniz International Proceedings in Informatics



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

1 Introduction

The enumeration of objects that satisfy a given property has applications in many scientific domains including biology and artificial intelligence. Enumeration can also be used as part of an exact algorithm, e.g., confer the algorithm by Lawler [27] to compute a coloring of an input graph using a minimum number of colors. The dynamic programming scheme used by this algorithm needs all the maximal independent sets of the input graph. It is worth noting that the running time depends mainly on a bound on the number of maximal independent sets as well as on the running time of an algorithm that would produce all these sets.

Clearly, the number of outputs of an enumeration algorithm can be exponential in the size of the given input. It is the case for the number of maximal independent sets: there are graphs with $3^{n/3}$ such sets [30], where n is the number of vertices in the graph. The running time of enumeration algorithms can either be measured with respect to the size of the input plus the size of the outputted set of objects, which is called *output-sensitive* analysis, or it can be measured according to the size of the input only, being called *input-sensitive* analysis. In the latter, the running time upper bound often implies an upper bound on the number of enumerated objects, *i.e.*, the maximum number of objects that can fulfill the given property.

Given a graph $G = (V, E)$, the problem of computing a minimum dominating set asks for a smallest-cardinality subset $S \subseteq V$ such that each vertex not in S has at least one neighbor in S . This well studied NP-hard problem attracted considerable attention for decades. Several exponential-time algorithms have been designed to solve the problem exactly, and the most recent are based on *Measure-and-Conquer* techniques to analyze their running times [15, 24, 31]. The problem of enumerating all inclusion-minimal dominating sets has also caught attention for general graphs as well as for special graph classes [10, 16, 19].

Many variants of the dominating set problem have also gained attention [22]. In particular, the minimum *connected* dominating set problem requires that the graph induced by S is connected. The problem has attracted great attention and various methods have been devised to solve it exactly [3, 14]. A more challenging question has been posed about the enumeration of inclusion-minimal connected dominating sets. Already designing an algorithm faster than $\text{poly}(n)2^n$ is known to be challenging, and this specific question has been asked several times as an open problem [6, 13, 20]. A recent result by Lokshantov et al. [29] shows that minimal connected dominating sets can be enumerated in time $2^{(1-\epsilon)n} \cdot n^{\mathcal{O}(1)}$, which broke the 2^n -barrier for the first time. It is worth noting that ϵ is a tiny constant, around 10^{-50} , and it has remained open whether an algorithm exists that can substantially break the 2^n -barrier. The enumeration of minimal connected dominating sets also received notable interest when the input is restricted to special graph classes [20, 21, 32, 33, 34].

On the other hand, the maximum number of minimal CDS in a graph was shown to be in $\Omega(3^{\frac{n}{3}})$ [20], which is obviously very low compared to the currently best upper bound. This gap between upper and lower bounds is narrower when it comes to special graph classes. On chordal graphs, for example, the upper bound has been recently improved to $\mathcal{O}(1.4736^n)$ [21]. Other improved lower/upper bounds have been obtained for AT-free, strongly chordal, distance-hereditary graphs, and cographs in [20]. Further improved bounds for split graphs, cobipartite and convex bipartite graphs have been obtained in [34] and [33]. Moreover, although the optimization problem seems simpler, the best-known exact algorithm solves the problem in time $\mathcal{O}(1.8619^n)$ [3]. This is already much larger than the best-known lower bounds of $3^{(n-2)/3}$ [20] to enumerate all minimal connected dominated sets.

In this paper, we show that the enumeration of all inclusion-wise minimal connected dominating sets can be achieved in time $\mathcal{O}(1.9896^n)$. Surprisingly, achieving this improvement was simply based on first considering the same enumeration on 2-degenerate graphs and

proving it to be possible in time $\mathcal{O}(1.9767^n)$. Achieving enumeration with polynomial delay is believed to be hard, since it would also lead to the same for the enumeration of minimal dominating sets, which has been open for several decades. We give further evidence of this (possible) hardness by showing that extending a subset of vertices into a minimal connected dominating set is NP-complete and also hard in a natural parameterized setting. Furthermore, we narrow the gap between upper and lower bounds by showing that the maximum number of minimal connected dominating sets in a graph is in $\Omega(1.4890^n)$, thus improving the previous lower bound of $\Omega(1.4422^n)$. Our construction yields new lower bounds on several special graph classes such as 3-degenerate planar bipartite graphs. For space restrictions, most proofs are either omitted or reduced to proof sketches; full proofs can be found in the long version of this paper [1].

2 Definitions, Preliminaries and Summary of Main Results

In this paper, we deal with undirected simple finite graphs that can be specified as $G = (V, E)$, where V is the finite vertex set and $E \subseteq \binom{V}{2}$ is the set of edges. The number of vertices $|V|$ is also called the *order* of graph G and is denoted by n . An edge $\{u, v\}$ is usually written as uv . Alternatively, E can be viewed as a symmetric binary relation, so that E^* is then the transitive closure of E , which is an equivalence relation whose equivalence classes are also known as *connected components*. A graph is called *connected* if it has only one connected component. A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$; G' is a *partial graph* of G if $V = V'$. A set of vertices S *induces* the subgraph $G[S] = (S, E_S)$, where $E_S = \{uv \in E \mid u, v \in S\}$; S is called *connected* if $G[S]$ is connected. For a vertex $v \in V$, $N_G(v) = \{u \in V \mid uv \in E\}$ is the *open neighborhood* of v , collecting the vertices *adjacent* to v ; its cardinality $|N_G(v)|$ is also called the *degree* of v , denoted as $\deg_G(v)$. We denote the *closed neighborhood* of v by $N_G[v] = N_G(v) \cup \{v\}$. We can extend set-valued functions to set arguments; for instance, $N_G[S] = \bigcup_{v \in S} N_G[v]$ for a set of vertices S ; S is a *dominating set* if $N_G[S] = V$. Whenever clear from context, we may drop the subscript G from our notation. If $X \subseteq V$, we also write $N_X(v)$ instead of $N(v) \cap X$ and $\deg_X(v)$ for $|N(v) \cap X|$. For brevity, we write CDS for *connected dominating set*. Next, we collect some observations.

► **Observation 1.** *If S is a CDS of a partial graph G' of G , then S is a CDS of G .*

Proof. We can think of $G = (V, E)$ as being obtained from G' by adding edges. Hence, if $N_{G'}[S] = V$, then $N_G[S] = V$. Moreover, adding edges cannot violate connectivity. ◀

► **Corollary 2.1.** *Let S be a CDS both of G and of a partial graph G' of G . If S is a minimal CDS of G , then it is a minimal CDS of G' .*

A graph $G = (V, E)$ is *d-degenerate* if there exists an *elimination ordering* (v_1, \dots, v_n) , where $V = \{v_1, \dots, v_n\}$, such that, for all $i = 1, \dots, n$, $\deg_{G[\{v_i, \dots, v_n\}]}(v_i) \leq d$. In other words, we can subsequently delete v_1, v_2, \dots from G , and at the time when v_i is deleted, it has degree bounded by d in the remaining graph. The decision problem CONNECTED DOMINATING SET EXTENSION expects as inputs a graph $G = (V, E)$ and a vertex set U , and the question is if there exists a minimal CDS S that extends U , *i.e.*, for which $S \supseteq U$ holds.

In the next section, we develop a branching algorithm. It is classical to analyze its running-time by solving recurrences of type $T(\mu(\mathcal{I})) = \sum_{i=1}^t T(\mu(\mathcal{I}) - r_i)$. Here, $\mu(\mathcal{I})$ is a measure on the size of the instance. The value t is the number of recursive calls (t is equal to 1 for *reduction rules*) and each r_i is (a lower bound on) the reduction of the measure corresponding to the recursive call. We simply denote by (r_1, r_2, \dots, r_t) the *branching vector* of the recurrence. We refer to the book by Fomin and Kratsch for further details on this standard analysis [17].

As discussed in the introduction, we shall first prove that all minimal CDS can be enumerated in time $\mathcal{O}(1.9767^n)$ on 2-degenerate graphs. This result is the key to our enumeration result for general graphs. This is why we will first present the corresponding branching enumeration algorithm for 2-degenerate graphs in a simplified form and analyze it with a rather simple measure in order to explain its main ingredients, and only thereafter, we turn towards a refined analysis that finally leads to the claimed enumeration result on general graphs. We shall prove that our input-sensitive enumeration algorithm cannot be turned into an enumeration algorithm with polynomial delay by simply interleaving the branching with tests for extendibility. For possible applications of such extension algorithms, we refer to the discussions in [7].

3 A CDS enumeration algorithm for 2-degenerate graphs

We are going to present an algorithm that enumerates all inclusion-wise minimal connected dominating sets (CDS) of a 2-degenerate graph $G = (V, E)$. Based on G , in the course of our algorithm, an instance is specified by $\mathcal{I} = (V'; O_d, O_n; S)$, consisting of four vertex sets that partition V . In the beginning, $\mathcal{I} = (V; \emptyset; \emptyset; \emptyset)$. In general, V' collects the vertices not yet decided by branching or reduction rules, $O := O_d \cup O_n$ are the vertices that have been decided *not* to be put into the solution, while S is the set of vertices decided to go into the solution that is constructed by the branching algorithm. The set O is further refined into O_n , the set of vertices that are not yet dominated, *i.e.*, if $x \in O_n$, then $\deg_S(x) = 0$, and O_d , the set of vertices that are already dominated, *i.e.*, if $x \in O_d$, then $\deg_S(x) > 0$. Similarly, we will sometimes refine $V' = V'_d \cup V'_n$. Notice that also vertices known to be dominated could be still put in the dominating set, either to dominate other vertices or for connectivity purposes. In the leaves of the branching tree, only instances of the form $(\emptyset, O_d, \emptyset, S)$ are of interest. Yet, before outputting S as a solution, one has to check if S is connected and if it does not contain a smaller CDS.

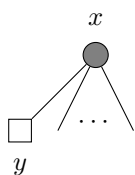
The algorithm actually starts by creating n different branches, in each a single vertex is put in S as a starting point, so that S is never empty. More precisely, the n^{th} branch would decide *not* to put the previously considered $n - 1$ vertices into the solution but the n^{th} one is put into S . This binary branching avoids generating solutions twice. Also, it is trivial to check in each branch if the selected vertex already dominates the whole graph, so that we can henceforth assume that G cannot be dominated by a single vertex.

We denote by c the number of connected components of $G[S]$. Now, we are ready to define the measure that we use to analyze the running time of our algorithm, following a very simple version of the *measure-and-conquer*-paradigm, as explained in [15, 17],

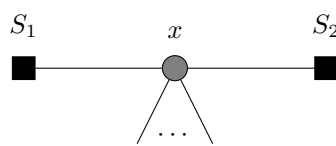
$$\mu(\mathcal{I}) = |V'| + \alpha \cdot |O_n| + \delta \cdot c.$$

We decide that $0 < \alpha, \delta < 1$, but we will determine the concrete values later as to minimize the upper-bound on the running-time. At the beginning, $V' = V$, $O_n = \emptyset$ and $c = 0$, so that then the measure equals $|V|$. At the end, $V' = O_n = \emptyset$ and the measure equals δ if the solution is connected and is bigger than δ if the solution is not connected.

For the possible branchings, we only consider vertices in a partial graph G' of $G[V' \cup O_n]$. As G is 2-degenerate, G' is also 2-degenerate, so that we can always find a vertex of degree at most two in G' . Some of our branchings apply to vertices of arbitrary degree, though; in such a situation, we denote the vertex that we branch on as x . If we branch on a small-degree vertex (due to 2-degeneracy), this vertex is called u . Clearly, a binary branch that puts a selected vertex either into S or into O is a complete case distinction.



(a) $x \in V'_d$ has neighbors in O_n .



(b) Vertices from two different connected components S_1, S_2 of $G[S]$ dominate x in G .

■ **Figure 1** Simple branchings for dominated vertices: Rules 1 and 2.

We are now explaining the conventions that we follow in our illustrations of subgraphs of G' . Vertices in V' are depicted by \ominus and more specifically by \circ if in V'_n or by \bullet if in V'_d . We use black squares \blacksquare to depict vertices which are already decided to belong to the solution S . We use \square for vertices from O_n . So, circles are used for undecided vertices (these vertices might still be added to S), whereas squares are used for vertices being already decided (to belong to the solution or to be discarded). Vertices in $V' \cup O$ are depicted as half-filled diamonds \blacklozenge , and if the vertex is from $V'_n \cup O_n$, then we use an unfilled diamond \diamond to represent it. A dashed line indicates an edge that may be present. It should be clear that one could always move to the graph where vertices in S that belong to the same connected component in $G[S]$ are merged. In order to avoid drawing too many vertices from S in our pictures, we assume these mergings to have been performed. Hence, when we draw two vertices from S , they belong to different connected components.

In the following, the branching and reduction rules require to be executed in order, so that our instance will (automatically) satisfy some structural properties when we apply one of the later rules. We can separate our branching and reduction rules into three parts:

- A first set of rules (Branching Rules 1, 2, 3, 4, Reduction Rules 1, 2, 3, 5) that deals with vertices of arbitrary degree that are (possibly) dominated, but only in some special cases, as detailed later. Those rules are applied first, so that whenever we apply a rule from the next two sets, we know that any dominated vertex is dominated by vertices from exactly one connected component of $G[S]$ and none of the vertices in its neighborhood are dominated. This means that the set of vertices V'_d as well as the set of vertices O_n forms an independent set in G' , a partial graph of $G[V' \cup O_n]$.
- A second set of rules (Branching Rules 5, 6, Reduction Rule 4) that handles the cases where the small-degree vertex that exists by the 2-degeneracy is undominated. If we apply a rule from the third set, every undominated vertex is of degree at least 3.
- The last set of rules (Branching Rules 7, 8, 9, 10) handles only the cases where the small-degree vertex that exists by the 2-degeneracy is dominated.

Note that even inside those three sets, rules have to be executed in the given order.

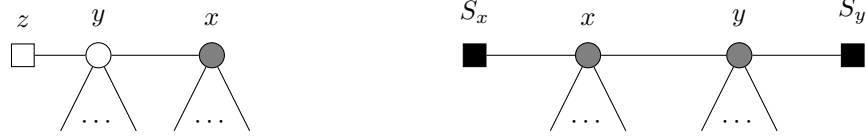
► **Branching Rule 1.** Let $x \in V'_d$ with $\deg_{O_n}(x) \geq 1$ (Figure 1a). Then branch as follows.

- (1) Put x in O_d .
- (2) Put x in S and every vertex in $N_{O_n}(x)$ in O_d .

► **Lemma 3.1.** *The branching of rule 1 is a complete case distinction. Moreover, it leads to a branching vector that is not worse than $(1, 1 + \alpha)$.*

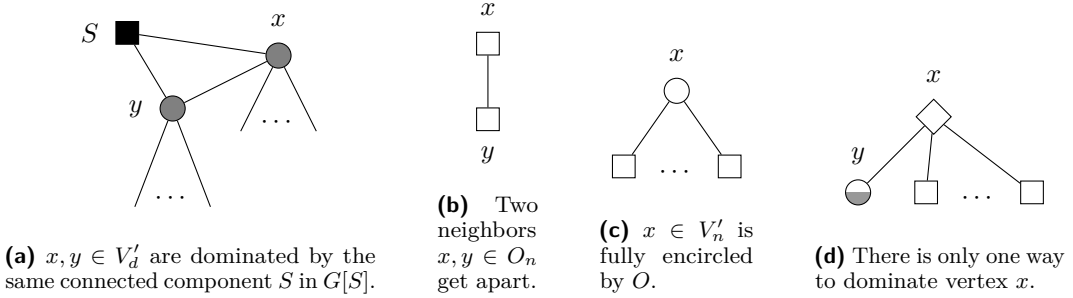
Proof. As $\deg_S(x) \geq 1$, we find that, if x is not put into the solution, then it is put into O_d , which decreases the measure by 1 in the first component of the branching vector. If x is put into the solution, then its neighbors are dominated and we decrease the measure by at least $1 + \alpha$ in the second component of the branching vector, as $\deg_{O_n}(x) \geq 1$. ◀

1:6 Enumerating Minimal Connected Dominating Sets



(a) A vertex from O_n in the second neighborhood of $x \in V'_d$ gives still an advantage. (b) $N_S(x)$ belongs to the same connected component S_x of $G[S]$, and so does $N_S(y)$ belong to S_y , but $S_x \neq S_y$.

■ Figure 2 Branching Rules 3 and 4.



(a) $x, y \in V'_d$ are dominated by the same connected component S in $G[S]$. (b) Two neighbors $x, y \in O_n$ get apart. (c) $x \in V'_n$ is fully encircled by O . (d) There is only one way to dominate vertex x .

■ Figure 3 Illustrating Reduction Rules 1, 3, 4 and 5.

We will have similar lemmas for each of the branching rules; we will summarize them at the end of this section instead of formulating them separately and point to the long version of the paper.

► **Branching Rule 2.** Let $x \in V'_d$ such that x is adjacent to two different connected components of $G[S]$; see Figure 1b. Then branch as follows. (1) Put x in O_d . (2) Put x in S .

We are now presenting two branching rules that could be viewed as variations of the first two; they always give worse branchings.

► **Branching Rule 3.** Let $x \in V'_d, y \in N_{V'_n}(x), z \in N_{O_n}(y)$ (Figure 2a). Then branch as follows. (1) Put x in O_d . (2) Put x in S, y in O_d . (3) Put x, y in S and thus $z \in O_d$.

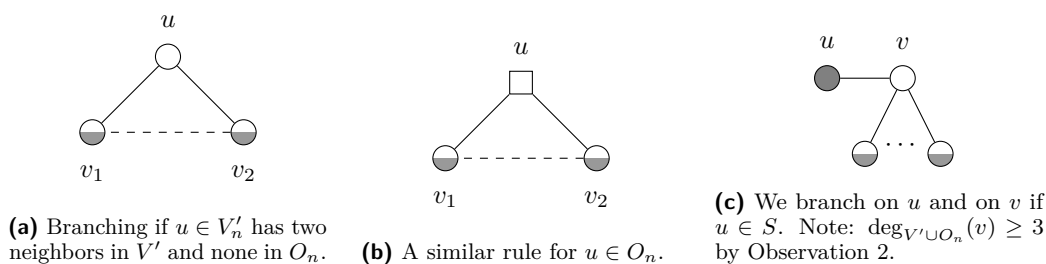
► **Branching Rule 4.** Let $x, y \in V'_d, xy \in E$, such that $z \in \{x, y\}$ is adjacent to a connected component S_z of $G[S]$, with $S_x \neq S_y$, see Figure 2b. Then branch as follows. (1) Put x in O_d . (2) Put x in S and y in O_d . (3) Put x in S and y in S .

Notice that in the last branch, the number of connected components decreases.

► **Reduction Rule 1.** If $x, y \in V'_d$ and $xy \in E$, then delete the edge xy ; see Figure 3a.

► **Lemma 3.2.** *Reduction Rule 1 is sound and the measure does not change.*

Proof. Let M be a minimal CDS of G such that $M \setminus V' = S$. Define $e = xy$ and $\tilde{G} = (V, E \setminus \{e\})$. Now we want to show that M is also a minimal CDS in \tilde{G} . Since x and y are already dominated by S , the deletion of the edge e would not affect domination, nor could x ever be the private neighbor of y or vice versa. The connectivity is only important if $x, y \in M$. Vertices x, y are dominated by the same connected component of S , as otherwise Branching Rule 4 would have applied with priority. Hence, there exists a path $p = (x, p_1, \dots, p_l, y)$, with internal vertices in S . Let $q = (q_1, \dots, q_k)$ be a path in $G[M]$ such that there exists an $i \in \{1, \dots, k-1\}$ with $q_i = x$ and $q_{i+1} = y$. Then $\tilde{q} = (q_1, \dots, q_i, p_1, \dots, p_l, q_{i+1}, \dots, q_k)$ is a walk in $\tilde{G}[M]$. Thus, $\tilde{G}[M]$ is connected and M is a CDS of \tilde{G} . As \tilde{G} is a partial graph, M is also a minimal CDS of G by Corollary 2.1. ◀



■ **Figure 4** Branching Rules 5, 6 and 7.

We can formulate and prove similar lemmas for the following reduction rules, as well. We will summarize this in one single lemma below instead.

► **Reduction Rule 2.** If x is an isolated vertex in $G[V' \cup O_n]$, do the following:

- If x is dominated, put x into O_d .
- If x is not dominated, skip this branch. (This will always happen if $x \in O_n$.)

► **Reduction Rule 3.** Let $x, y \in O_n$ be with $xy \in E$; see Figure 3b. We delete the edge xy .

► **Reduction Rule 4.** If $x \in V'_n$ obeys $N(x) \cap V' = \emptyset$, then skip this branch (Figure 3c).

The next rule will even decrease the measure by at least $1 - \delta$.

► **Reduction Rule 5.** Let $x \in V'_n \cup O_n$, with $N_{V'}(x) = \{y\}$. Then, put y into S .

► **Branching Rule 5.** Let $u \in V'_n$ with $\deg_{V'}(u) = 2$, $\deg_{O_n}(u) = 0$ and $N_{V'}(u) = \{v_1, v_2\}$; see Figure 4a. Then branch as follows. (1) Put u in O_d , v_1 in S . (2) Put u in O_d , v_1 in O , v_2 in S . (3) Put u in S , v_1 in S . (4) Put u in S , v_1 in O_d , v_2 in S .

More precisely, in the second branch, we put v_1 into O_d if $v_1v_2 \in E$ or if v_1 was already dominated and we put v_1 into O_n , otherwise. The same is done in the Branching Rules 6, 7, 8, 9 and 10, as it can be decided whether the vertex goes into O_n or into O_d .

► **Branching Rule 6.** Let $u \in O_n$ with $N_{V'}(u) = \{v_1, v_2\}$ (Figure 4a). Then branch as follows. (1) Put v_1 in S , and thus u in O_d . (2) Put v_1 in O , v_2 in S , and thus u in O_d .

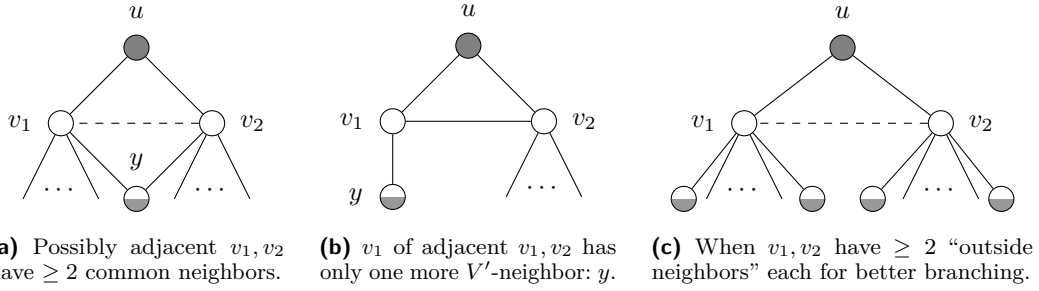
► **Observation 2.** For each rule below, as it is applied in particular after Branching Rule 5 and Reduction Rule 5, we observe: for any vertex $v \in V'_n \cup O_n$, we have $\deg_{V' \cup O_n}(v) \geq 3$.

► **Branching Rule 7.** Let $u \in V'_d$ with $N_{V'}(u) = \{v\}$ and $\deg_{O_n}(v) = 0$; see Figure 4c. Then branch as follows. (1) Put u in O_d . (2) Put u in S and v in O_d and thus all the vertices of $N_{V'}(v) \setminus \{u\}$ in O . (3) Put u in S and v in S .

► **Branching Rule 8.** Let $u \in V'_d$, with $N_{V'}(u) = \{v_1, v_2\}$ and $\deg_{O_n}(u) = 0$ (Figure 5a). If $N_{V'}(v_1) \cap N_{V'}(v_2)$ contains a vertex $y \in V'$ different from u , then branch as follows. (1) Put u in O_d . (2) Put u in S , v_1 in S . (3) Put u in S , v_1 in O_d , v_2 in S . (4) Put u in S , v_1 in O_d , v_2 in O_d and thus y in O .

► **Branching Rule 9.** Let $u \in V'_d$, $N_{V'}(u) = \{v_1, v_2\}$, $N_{V'_d}(u) = N_{O_n}(u) = \emptyset$ and $N_{V'}(v_1) \setminus \{u, v_2\} = \{y\}$; see Figure 5b. Then branch as follows. (1) Put u in O_d . (2) Put u in S , v_1 in S . (3) Put u in S , v_1 in O_d , v_2 in S . (4) Put u in S , v_1 in O_d , v_2 in O_d and y in O . (5) Put u in S , v_1 in O_d , v_2 in O_d and y in S and thus the vertices of $N_{V'}(v_2) \setminus \{u, v_1\}$ in O .

► **Branching Rule 10.** (Figure 5c) Let $u \in V'_d$, $N_{V'}(u) = \{v_1, v_2\}$, $N_{V'_d}(u) = N_{O_n}(u) = \emptyset$, such that $|N_{V'}(v_1) \setminus \{u, v_2\}| \geq 2$, as well as $|N_{V'}(v_2) \setminus \{u, v_1\}| \geq 2$. Then branch as follows. (1) Put u in O_d . (2) Put u in S , v_1 in S . (3) Put u in S , v_1 in O_d , v_2 in S . (4) Put u in S , v_1 in O_d , v_2 in O_d and all of $N_{V'}(v_1) \setminus \{u, v_2\}$ in O . (5) Put u in S , v_1 in O_d , v_2 in O_d and all of $N_{V'}(v_2) \setminus \{u, v_1\}$ in O .



■ **Figure 5** Branching Rules 8, 9 and 10.

The next and final rule will never be applied when this algorithm is applied to a 2-degenerate graph. It rather prepares the ground for the general case. However, with our current measure, this would yield an $\mathcal{O}(2^n)$ algorithm in the general case. With a modified measure, as described in the next section, we will achieve a better running time.

► **Branching Rule 11.** Let $x \in V'_d$. Then we branch as follows. (1) Put x in O_d . (2) Put x in S and thus the vertices of $N_{V'}(x)$ in V'_d .

► **Lemma 3.3.** *Each of the mentioned branching rules covers all cases in their described respective situation. The branching will lead to a branching vector as listed in Table 1.*

► **Lemma 3.4.** *The mentioned reduction rules are sound and their application never increases the measure.*

A case analysis shows the correctness of our algorithm in the following sense:

► **Lemma 3.5.** *The Reduction and Branching Rules cover all possible cases.*

Proof. As explained below, Branching Rule 11 serves as a final catch-all. For the 2-degenerate case, we should focus on all other rules and prove that they cover all cases. This means that we have to show that the proposed algorithm will resolve each 2-degenerate graph completely. Our rule priorities might remove vertices of arbitrary degree from the graph G' that is a partial graph of the graph $G[V' \cup O_n]$. This way, we again arrive at a 2-degenerate graph. Yet, what is important for dealing with 2-degenerate graphs is to consider all cases of a vertex u of degree at most 2. The degree conditions in the following case distinction refer to G' . Degree-0 vertices are treated with Reduction Rule 2. We now discuss vertices u of degrees one or two. There are two different cases: either u is in $V'_n \cup O_n$, or $u \in V'_d$.

Case 1: u is not dominated by S . Now we discuss its degree in G' , either it is 1 or it is 2.

Case 1.1: $\deg(u) = 1$ and we denote by v the neighbor in G' of u . If $v \in O_n$, then either u is in O_n and then satisfies the conditions of Reduction Rule 3, or it is in V_n and thus satisfies the conditions of Reduction Rule 4. If $v \notin O_n$, v satisfies the conditions of Reduction Rule 5.

Case 1.2: $\deg(u) = 2$. Now we discuss the number of neighbors of u in O_n . If both neighbors are in O_n , Reduction Rule 4 applies if $u \in V'_n$ and Reduction Rule 3 applies otherwise. If only one neighbor of u is in O_n , then this means that if $u \in O_n$ Reduction Rule 3 applies, and otherwise Reduction Rule 5 applies. If none of the neighbors are in O_n , then either $u \in V'_n$ and then Branching Rule 5 applies, or otherwise $u \in O_n$ and thus Branching Rule 6 applies.

Case 2: u is dominated by S .

■ **Table 1** Collection of all branching vectors for the 2-degenerate case; the branching numbers are displayed for the different cases with $\alpha = 0.106$ and $\delta = 0.106$.

Branching Rule #	Branching vector	Branching number
1	$(1, 1 + \alpha)$	always better than 3
2	$(1, 1 + \delta)$	always better than 4
3	$(1, 2, 2 + \alpha)$	< 1.9766
4	$(1, 2, 2 + \delta)$	< 1.9766
5	$(2 - \delta, 3 - \delta - \alpha, 2 - \delta, 3 - \delta)$	< 1.8269
6	$(1 + \alpha - \delta, 2 - \delta)$	< 1.6420
7	$(1, 4 - 2\alpha, 2)$	< 1.7691
8	$(1, 2, 3, 4 - \alpha)$	< 1.9333
9	$(1, 2, 3, 4 - \alpha, 5 - \delta - \alpha)$	< 1.9767
10	$(1, 2, 3, 5 - 2\alpha, 5 - 2\alpha)$	< 1.9420
11	$(\beta, 3 - 2\beta)$ where $\beta = 1$	$= 2$ not for 2-degenerate graphs

Case 2.1: $\deg(u) = 1$. We denote by v the neighbor in G' of u . If v is dominated by S then Reduction Rule 1 applies. If v is not dominated by S , then either it has no neighbors in O_n and then Branching Rule 7 applies, or it has at least one neighbor in O_n and Branching Rule 3 applies.

Case 2.2: $\deg(u) = 2$. We denote $\{v_1, v_2\} = N_{V'}(u)$. If either v_1 or v_2 is dominated, then Reduction Rule 1 applies. If both of them are not dominated by S , either they have at least one neighbor in O_n and then Branching Rule 3 applies or they do not have any neighbor in O_n . So we know that $\deg_{V'}(v_1) \geq 3$, and $\deg_{V'}(v_2) \geq 3$, otherwise we could apply one of the rules of the previous case to v_1 or v_2 . Now, either v_1 and v_2 have a common neighbor outside of u and Branching Rule 8 applies, or they do not and then either at least one of them has exactly one neighbor that is not u or v_1 (or v_2 , respectively) and Branching Rule 9 applies, or they both have at least two neighbors outside of u, v_1 and v_2 , so that Branching Rule 10 applies.

We finally have to prove the correctness of the algorithm for a general graph. If any vertex satisfies the conditions of any Branching or Reduction Rule apart from Branching Rule 11, then we apply such a rule, otherwise no such rule applies, which means that the minimum degree of $G'[V' \cup O_n]$ is 3. If at least one vertex is in V_d , then we can apply Branching Rule 11. Assume it is not the case, that means V_d is empty, so every vertex of $V' \cup O_n$ is not dominated (and it is not empty, otherwise the algorithm would have ended). Since in the beginning, S was not empty, S is never empty. Moreover, at any point in the algorithm, $N(S) = V_d \cup O_d$. So in our case, we have $N(S) = O_d$. This means that S is not a dominating set (there are vertices in $V_n \cup O_n$) and S cannot have any neighbor added, so there is no CDS $M \subset V$ such that $M \setminus V' = S$. This branch has to be discarded. ◀

► **Theorem 3.6.** *CONNECTED DOMINATING SET ENUMERATION can be solved in time $\mathcal{O}(1.9767^n)$ on 2-degenerate graphs of order n , using polynomial space only. The claimed branching number is attained by setting $\alpha = 0.106$ and $\delta = 0.106$; see Table 1.*

► **Corollary 3.7.** *2-degenerate graphs have no more than $\mathcal{O}(1.9767^n)$ minimal CDS.*

► **Remark 3.8.** We could deduce a corresponding CDS enumeration result for subcubic graphs, as they can be dealt with as 2-degenerate graphs after the initial branching. There is a clear indication that the bound that we derive in this way is not tight. Namely, Kangas et al. have shown in [25] that there are no more than 1.9351^n many connected sets of vertices in a subcubic graph of order n .

4 Getting β into the game: the general case

As indicated above, we are now refining the previous analysis by splitting the set of vertices of the currently considered graph further. More precisely, the set of vertices V' that have not been decided to come into or to be out of the solution S is split into the set of vertices V'_n that is still undominated and the set V'_d of vertices that is already dominated. From the viewpoint of the original graph, the neighbors of the solution set S are therefore partitioned into the sets V'_d and O_d . However, observe that although we do not consider O_d anymore in the present graph, we do care about V'_d , since V'_d -vertices can still be either placed into S or into O_d by future branching or reduction rules. This is also reflected in the measure of the instance I , which is now defined as:

$$\mu(I) = |V'_n| + \alpha \cdot |O_n| + \beta \cdot |V'_d| + \delta \cdot c$$

We set $\alpha = 0.110901$, $\beta = 0.984405$ and $\delta = 0.143516$. We are next working through the branching rules one by one, as in particular the branching vectors will now split off into different cases, as in our preliminary analysis, we only took care of the worst cases, not differentiating between V'_n or V'_d (which was summarized under the set name V' before). For the convenience of the reader, we repeat the formulation of the branching rules in the following, adapting the notations.

We start with a table stating the branching vectors according to this new measure for some branching rules for which it is straightforward. Branching Rule 3 and Branching Rule 4 are the tight cases in our Measure-and-Conquer analysis.

Rule	Branching vector	Br. number	Rule	Branching vector	Br. number
# 1	$(\beta, \beta + \alpha)$	< 1.9489	# 2	$(\beta, \beta + \delta)$	< 1.9297
# 3	$(\beta, \beta + 1, \beta + 1 + \alpha)$	< 1.9896	# 4	$(\beta, 2\beta, 2\beta + \delta)$	< 1.9896

Branching Rule 5 Let $u \in V'_n$ with $\deg_{V'}(u) = 2$, $\deg_{O_n}(u) = 0$ and $N_{V'}(u) = \{v_1, v_2\}$. The branching vector is different when $v_1, v_2 \in V'_d$ or $v_1, v_2 \in V'_n$. We will assume $v_1 v_2 \notin E$, as this is always the worst case. A similar analysis applies to Branching Rule 6.

Condition	Branching vector for # 5	Br. number	Br. vector for # 6	Br. number
$v_1, v_2 \in V'_n$	$(2 - \delta, 3 - \delta - \alpha, 2 - \delta, 3 - \delta)$	< 1.8463	$(1 + \alpha - \delta, 2 - \delta)$	< 1.6635
$v_1 \in V'_n, v_2 \in V'_d$	$(2 - \delta, 2 - \alpha + \beta, 2 - \delta, 2 + \beta)$	< 1.8236	$(1 + \alpha, 1 + \beta)$	< 1.5855
$v_1, v_2 \in V'_d$	$(1 + \beta, 1 + 2\beta, 1 + \beta, 1 + 2\beta)$	< 1.7785	$(\beta + \alpha, 2\beta + \alpha)$	< 1.5817

Branching Rule 7 We look at $u \in V'_d$, $\{v\} = V'_n \cap N(u)$ and the neighbors of v ; we know that $N(v)$ contains at least $u, v_1, v_2 \notin O$; differentiating between $v_i \in V'_n$ or $v_i \in V'_d$ (as before), we arrive at three cases never leading to a branching number worse than 1.78.

Branching Rule 8 Let $u \in V'_d$ be with $\deg_{V' \cup O_n}(u) = 2$, such that $v_1, v_2 \in N_{V'}(u)$ and $N_{V'}(v_1) \cap N_{V'}(v_2)$ contains a vertex $y \in V'$ different from u . We differentiate $y \in V'_n$ or $y \in V'_d$ and arrive at a branching number not worse than 1.9403.

Branching Rule 9 Let $u \in V'_d$ be with $\deg_{V' \cup O_n}(u) = 2$, such that $v_1, v_2 \in N_{V'}(u)$ and $N_{V'}(v_1) \setminus \{u, v_2\} = \{y\}$. We distinguish $y \in V'_n$ and $y \in V'_d$; the first case leads to another tight-case branching for our algorithm. In the last branch of each vector, $\min(\beta, 1 - \alpha)$ corresponds to whether $z \in N(v_2) \setminus \{u, v_1\}$ belongs to V'_d or to V'_n .

Condition	Branching vector	Branching number
$y \in V'_n$	$(\beta, 1 + \beta, 2 + \beta, 3 + \beta - \alpha, 3 + \beta - \delta + \min(\beta, 1 - \alpha))$	< 1.9896
$y \in V'_d$	$(\beta, 1 + \beta, 2 + \beta, 2 + 2\beta, 2 + 2\beta + \min(\beta, 1 - \alpha))$	< 1.9813

Branching Rule 10 Let $u \in V'_d$, $N_{V'_n}(u) = \{v_1, v_2\}$, $N_{V'_d}(u) = N_{O_n}(u) = \emptyset$. We further assume that $|N_{V'}(v_1) \setminus \{u, v_2\}| \geq 2$, as well as $|N_{V'}(v_2) \setminus \{u, v_1\}| \geq 2$. The only thing to discuss here is whether the vertices of $N_{V'}(v_1) \setminus \{u, v_2\}$, $N_{V'}(v_2) \setminus \{u, v_1\}$ are in V'_n or V'_d . This leads to nine subcases, which never produce a branching vector worse than 1.9453.

Branching Rule 11 When this branching rule applies, the sets in which the different vertices reside are all already known: as it is applied after Branching Rule 1 and Branching Rule 4, all the neighbors of $x \in V'_d$ are in V'_n , moreover, $\deg_{V' \cup O_n}(x) = \deg_{V'_n}(x) \geq 3$, as it is applied after every other rule. The branching vector $(\beta, 3 - 2\beta)$ gives a branching number < 1.9896 , which is the last tight-case branching.

► **Theorem 4.1.** *CONNECTED DOMINATING SET ENUMERATION can be solved in time $\mathcal{O}(1.9896^n)$ on graphs of order n , using polynomial space only.*

► **Corollary 4.2.** *There are no more than $\mathcal{O}(1.9896^n)$ many minimal connected dominating sets in a graph of order n .*

5 Achieving polynomial delay is not easy

How could we achieve polynomial delay for enumeration problems? If CONNECTED DOMINATING SET EXTENSION would be solvable in polynomial time, then we might cut search tree branches whenever it is clear that no solution is to be expected beyond a certain node of the search tree, as we can associate to such a node also a set of vertices U that is decided to go into the solution. For example, it has been recently exemplified with the enumeration problem of minimal Roman dominating functions in [2] that a polynomial-time algorithm for the corresponding extension problem can be adapted so that polynomial delay can be achieved for this type of enumeration problem. However, we can show that CONNECTED DOMINATING SET EXTENSION is NP-complete by a reduction from 3-SAT. This means that new ideas would be needed for showing polynomial delay for enumerating minimal connected dominating sets.

► **Theorem 5.1.** *The CONNECTED DOMINATING SET EXTENSION problem is NP-complete, even when restricted to 2-degenerate graphs.*

Due to the parsimonious nature of the reduction, we can also conclude the following result.

► **Corollary 5.2.** *Assuming that the Exponential Time Hypothesis¹ holds, there is no algorithm that solves the CONNECTED DOMINATING SET EXTENSION problem in time $\mathcal{O}(2^{o(n)})$ on (2-degenerate) graphs of order n .*

Hence, even any subexponential delay seems to be hard to achieve.

Furthermore, the CONNECTED DOMINATING SET EXTENSION with the *standard parameterization* $|U|$, where U is the given subset of V , is even W[3]-hard on split graphs. To show this, we need the W[3]-completeness of HITTING SET EXTENSION [4, 5, 7] with standard parameterization; in this problem, the input consists of a hypergraph (X, \mathcal{S}) , with $\mathcal{S} \subseteq 2^X$, and a set $U \subseteq X$, and the question if there exists a minimal hitting set H (this means that $H \cap S \neq \emptyset$ for all $S \in \mathcal{S}$) that extends U , i.e., $U \subseteq H$.

► **Theorem 5.3.** *CONNECTED DOMINATING SET EXTENSION (with standard parameterization) is W[3]-hard, even on split graphs.*

¹ For a discussion of this hypothesis, we refer to [23, 28].

On split graphs, we can also prove W[3]-completeness of CONNECTED DOMINATING SET EXTENSION. Few natural parameterized problems are known to be W[3]-complete [4, 5, 7, 8]. There is yet another stroke against hopes for obtaining a polynomial-delay enumeration algorithm for minimal CDS. Namely, we could see the reduction presented for Theorem 5.3 also as a reduction that proves the following statement, which connects our enumeration problem to HITTING SET TRANSVERSAL, a problem notoriously open for decades.

► **Theorem 5.4.** *If there was an algorithm for enumerating minimal CDS with polynomial delay in split graphs, then there would be an algorithm for enumerating minimal hitting sets in hypergraphs with polynomial delay.*

6 Lower bounds

Several attempts to construct lower bound examples are known from the literature, leading to $3^{(n-2)/3} \in \Omega(1.4422^n)$ many minimal connected dominating sets in n -vertex graphs [20, 33]. We now present an improved lower bound on the maximum number of minimal connected dominating sets in a graph. Given arbitrary positive integers k, t , we construct a graph G_t^k of order $n = k(2t + 1) + 1$ as follows. The main building blocks of G_t^k consist of k copies of a base-graph G_t , of order $2t - 1$. The vertex set of G_t consists of three layers. The first one is a set $X = \{x_1, \dots, x_t\}$ that induces a clique. The second one is an independent set $Y = \{y_1, \dots, y_t\}$, while the third layer consists of a singleton $\{z\}$. Each vertex $x_i \in X$ has exactly $t - 1$ neighbors in Y : $N(x_i) = \{y_j \in Y : i \neq j\}$. Hence, $X \cup Y$ induces a copy of $K_{t,t}$ minus a perfect matching. Finally the vertex z is adjacent to all vertices in Y . Figure 6a shows the graph G_t for $t = 4$. To finally construct the graph G_t^k , we introduce a final vertex s that is connected to all vertices of each set X of each copy of the base-graph.

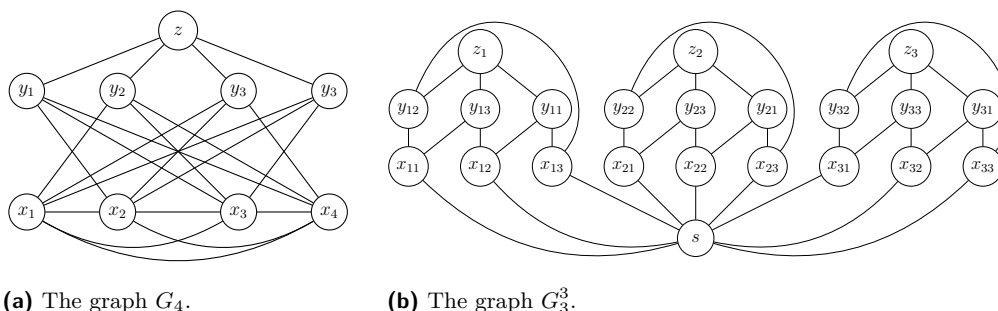
► **Lemma 6.1.** *For each $t > 0$, the graph G_t has exactly $\frac{t^3+t^2}{2} - t$ minimal connected dominating sets that have non-empty intersection with the set X .*

Proof. The set X cannot have more than two vertices in common with any minimal CDS, since any two elements of X dominate $X \cup Y$. Any minimal CDS that contains exactly one vertex x_i of X must contain the vertex z , to dominate y_i , and one of the $t - 1$ neighbors of x_i (to be connected). There are $t(t - 1)$ sets of this type. Moreover, each pair of elements of X dominates Y . So a minimal CDS can be formed by (any) two elements of X and any of the elements of Y (to dominate z). There are $t \frac{t(t-1)}{2}$ such sets. ◀

The hub-vertex s in G_t^k must be in any CDS, being a cut-vertex. Therefore, there is no need for the set X in G_t to induce a clique, being always dominated by s . In other words, the counting used in the proof above still holds if each copy of G_t is replaced by $G_t - E(X)$ in G_t^k . Here, $E(X)$ denotes the set of edges in $G_t[X]$. Figure 6b shows G_3^3 without the edges between pairs of element of X in each copy of G_3 . With the help of Lemma 6.1, we can show:

► **Theorem 6.2.** *The maximum number of minimal CDS in a connected graph of order n is in $\Omega(1.4890^n)$; an example family of graphs is (G_4^k) .*

Proof. By Lemma 6.1, each copy of the graph G_t has $\frac{t^3+t^2}{2} - t$ minimal CDS. There are k such graphs in G_t^k , in addition to the vertex s that connects them all. Every minimal CDS must contain s and at least one element from $N(s)$ in each G_t . Therefore, the total number of minimal CDS in G_t^k is $(\frac{t^3+t^2}{2} - t)^k = (\frac{t^3+t^2}{2} - t)^{\frac{n-1}{2t+1}}$. The claimed lower bound is achieved when $t = 4$, which gives a total of $36^{\frac{n-1}{9}} \in \Omega(1.4890^n)$. ◀

(a) The graph G_4 .(b) The graph G_3^3 .

■ **Figure 6** How our lower bound examples are composed.

We note that G_t^k is a t -degenerate graph that is also bipartite (since the set X in each copy of G_t can be an independent set). Furthermore, G_3^k is planar, as intentionally drawn in Figure 6b. Our general formula (Lemma 6.1) yields that G_3^k has $15^{n/7} = \Omega(1.4723^n)$ minimal CDS, improving on the previously mentioned construction for 3-degenerate graphs in [33].

► **Corollary 6.3.** *The maximum number of minimal connected dominating sets in a 3-degenerate bipartite planar graph of order n is in $\Omega(1.4723^n)$.*

Finally, G_2^k is a 2-degenerate graph of order n with $4^{n/5} = \Omega(1.3195^n)$ many minimal CDS, incidentally matching the best known lower bound in cobipartite graphs [9].

7 Conclusions and Open Problems

In our paper, we focused on developing an input-sensitive enumeration algorithm for minimal CDS. We achieved some notable progress both on the running time for such enumeration algorithms and with respect to the lower bound examples. However, the gap between lower and upper bound is still quite big, and the natural question to ask here is to bring lower and upper bounds closer; in an optimal setting, both would match. We are working on a further refined version that will bring down the upper bound a bit, but not decisively. This question of non-matching upper and lower bounds is also open for most special graph classes.

One particular such graph class that is studied in this paper is the class of 2-degenerate graphs. We like to suggest to study this graph class also for other enumeration problems, or, more generally, for problems that involve a measure-and-conquer analysis of branching algorithms, because this was the key to break the 2^n -barrier significantly for enumerating minimal CDS with measure-and-conquer, something that seemed to be impossible with other more standard approaches, like putting weights to low-degree vertices.

As we also proved that the extension problem associated to CDS is computationally intractable even on 2-degenerate graphs, it is not that straightforward to analyze our enumeration algorithm with the eyes of output-sensitive analysis. Conversely, should it be possible to find an efficient algorithm for an extension problem, also on special graph classes, then usually polynomial-delay algorithms can be shown; as a recent example in the realm of domination problems, we refer to the enumeration of minimal Roman dominating functions described in [2]. So, in the context of our problem, we can ask: Can we achieve polynomial delay for any enumeration algorithm for minimal CDS? Can we combine this analysis with a good input-sensitive enumeration approach? Notice that the corresponding questions for

enumerating minimal dominating sets are an open question for decades. This is also known as the HITTING SET TRANSVERSAL problem; see [11, 12, 18, 26]. We therefore also presented relations between polynomial-delay enumeration of minimal dominating sets and that of minimal CDS, again explaining the difficulty of the latter question. Finally, we also briefly discussed the possibility of subexponential delay. We propose to discuss this question further also for other enumeration problems when polynomial delay is not achievable, as it might well be a practical solution to know that the delay time is substantially smaller than the time needed to enumerate all solutions, but not polynomial time.

We also discussed parameterized complexity aspects of CONNECTED DOMINATING SET EXTENSION, leaving $W[3]$ -membership an open question in the general case.

References

- 1 F. N. Abu-Khzam, H. Fernau, B. Gras, M. Liedloff, and K. Mann. Enumerating connected dominating sets. Technical report, Cornell University, ArXiv/CoRR, 2022. doi:10.48550/arXiv.2205.00086.
- 2 F. N. Abu-Khzam, H. Fernau, and K. Mann. Minimal Roman dominating functions: Extensions and enumeration. Technical Report 2204.04765, Cornell University, ArXiv/CoRR, 2022. To appear in the Proceedings of WG 2022. doi:10.48550/arXiv.2204.04765.
- 3 F. N. Abu-Khzam, A. E. Mouawad, and M. Liedloff. An exact algorithm for connected red-blue dominating set. *Journal of Discrete Algorithms*, 9(3):252–262, 2011.
- 4 T. Bläsius, T. Friedrich, J. Lischeid, K. Meeks, and M. Schirneck. Efficiently enumerating hitting sets of hypergraphs arising in data profiling. *Journal of Computer and System Sciences*, 124:192–213, 2022.
- 5 T. Bläsius, T. Friedrich, and M. Schirneck. The complexity of dependency detection and discovery in relational databases. *Theoretical Computer Science*, 900:79–96, 2022.
- 6 H. L. Bodlaender, E. Boros, P. Heggernes, and D. Kratsch. Open problems of the Lorentz workshop “Enumeration algorithms using structure”. Technical Report UU-CS-2015-016, Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, November 2015.
- 7 K. Casel, H. Fernau, M. Khosravian Ghadikolaei, J. Monnot, and F. Sikora. On the complexity of solution extension of optimization problems. *Theoretical Computer Science*, 904:48–65, 2022.
- 8 J. Chen and F. Zhang. On product covering in 3-tier supply chain models: Natural complete problems for $W[3]$ and $W[4]$. *Theoretical Computer Science*, 363(3):278–288, 2006.
- 9 J.-F. Couturier, P. Heggernes, P. van ’t Hof, and D. Kratsch. Minimal dominating sets in graph classes: Combinatorial bounds and enumeration. In M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science*, volume 7147 of *LNCS*, pages 202–213. Springer, 2012.
- 10 J.-F. Couturier, R. Letourneur, and M. Liedloff. On the number of minimal dominating sets on some graph classes. *Theoretical Computer Science*, 562:634–642, 2015.
- 11 N. Creignou, M. Kröll, R. Pichler, S. Skritek, and H. Vollmer. A complexity theory for hard enumeration problems. *Discrete Applied Mathematics*, 268:191–209, 2019.
- 12 T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.
- 13 H. Fernau, P. A. Golovach, and M.-F. Sagot. Algorithmic enumeration: Output-sensitive, input-sensitive, parameterized, approximative (Dagstuhl Seminar 18421). *Dagstuhl Reports*, 8(10):63–86, 2018.
- 14 F. V. Fomin, F. Grandoni, and D. Kratsch. Solving Connected Dominating Set faster than 2^n . *Algorithmica*, 52:153–166, 2008.

- 15 F. V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5), 2009.
- 16 F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms*, 5(1):1–17, 2008.
- 17 F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. Springer, 2010.
- 18 A. Gainer-Dewar and P. Vera-Licona. The minimal hitting set generation problem: Algorithms and computation. *SIAM Journal of Discrete Mathematics*, 31(1):63–100, 2017.
- 19 P. A. Golovach, P. Heggernes, M. Moustapha Kanté, D. Kratsch, and Y. Villanger. Minimal dominating sets in interval graphs and trees. *Discrete Applied Mathematics*, 216:162–170, 2017.
- 20 P. A. Golovach, P. Heggernes, and D. Kratsch. Enumerating minimal connected dominating sets in graphs of bounded chordality. *Theoretical Computer Science*, 630:63–75, 2016.
- 21 P. A. Golovach, P. Heggernes, D. Kratsch, and R. Saei. Enumeration of minimal connected dominating sets for chordal graphs. *Discrete Applied Mathematics*, 278:3–11, 2020.
- 22 T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs*, volume 208 of *Monographs and Textbooks in Pure and Applied Mathematics*. Marcel Dekker, 1998.
- 23 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
- 24 Y. Iwata. A faster algorithm for dominating set analyzed by the potential method. In D. Marx and P. Rossmanith, editors, *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011*, volume 7112 of *LNCS*, pages 41–54. Springer, 2012.
- 25 K. Kangas, P. Kaski, J. H. Korhonen, and M. Koivisto. On the number of connected sets in bounded degree graphs. *Electron. J. Comb.*, 25(4):P4.34, 2018.
- 26 M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal of Discrete Mathematics*, 28(4):1916–1929, 2014.
- 27 E. L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, 1976.
- 28 D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *EATCS Bulletin*, 105:41–72, 2011.
- 29 D. Lokshtanov, M. Pilipczuk, and S. Saurabh. Below all subsets for minimal connected dominating set. *SIAM Journal of Discrete Mathematics*, 32(3):2332–2345, 2018.
- 30 J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.
- 31 J. Nederlof, J. M. M. van Rooij, and T. C. van Dijk. Inclusion/exclusion meets measure and conquer. *Algorithmica*, 69(3):685–740, 2014.
- 32 M. Y. Sayadi. *Construction et analyse des algorithmes exacts et exponentiels: énumération input-sensitive. (Design and analysis of exact exponential algorithms: input-sensitive enumeration)*. PhD thesis, University of Lorraine, Nancy, France, 2019.
- 33 M. Y. Sayadi. On the maximum number of minimal connected dominating sets in convex bipartite graphs. Technical Report abs/1908.02174, Cornell University, ArXiv, 2019. [arXiv:1908.02174](https://arxiv.org/abs/1908.02174).
- 34 I. B. Skjærten. Faster enumeration of minimal connected dominating sets in split graphs. Master’s thesis, Department of Informatics, University of Bergen, Norway, June 2017.

Non-Adaptive Edge Counting and Sampling via Bipartite Independent Set Queries

Raghavendra Addanki ✉ 🏠 

Adobe Research, San Jose, CA, USA

Andrew McGregor ✉ 🏠

Manning College of Information and Computer Sciences,
University of Massachusetts Amherst, MA, USA

Cameron Musco ✉ 🏠

Manning College of Information and Computer Sciences,
University of Massachusetts Amherst, MA, USA

Abstract

We study the problem of estimating the number of edges in an n -vertex graph, accessed via the *Bipartite Independent Set* query model introduced by Beame et al. (TALG '20). In this model, each query returns a Boolean, indicating the existence of at least one edge between two specified sets of nodes. We present a *non-adaptive* algorithm that returns a $(1 \pm \epsilon)$ relative error approximation to the number of edges, with query complexity $\tilde{O}(\epsilon^{-5} \log^5 n)$, where $\tilde{O}(\cdot)$ hides $\text{poly}(\log \log n)$ dependencies. This is the first non-adaptive algorithm in this setting achieving $\text{poly}(1/\epsilon, \log n)$ query complexity. Prior work requires $\Omega(\log^2 n)$ rounds of adaptivity. We avoid this by taking a fundamentally different approach, inspired by work on single-pass streaming algorithms. Moreover, for constant ϵ , our query complexity significantly improves on the best known adaptive algorithm due to Bhattacharya et al. (STACS '22), which requires $O(\epsilon^{-2} \log^{11} n)$ queries. Building on our edge estimation result, we give the first non-adaptive algorithm for outputting a nearly uniformly sampled edge with query complexity $\tilde{O}(\epsilon^{-6} \log^6 n)$, improving on the works of Dell et al. (SODA '20) and Bhattacharya et al. (STACS '22), which require $\Omega(\log^3 n)$ rounds of adaptivity. Finally, as a consequence of our edge sampling algorithm, we obtain a $\tilde{O}(n \log^8 n)$ query algorithm for connectivity, using two rounds of adaptivity. This improves on a three-round algorithm of Assadi et al. (ESA '21) and is tight; there is no non-adaptive algorithm for connectivity making $o(n^2)$ queries.

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

Keywords and phrases sublinear graph algorithms, bipartite independent set queries, edge sampling and counting, graph connectivity, query adaptivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.2

Related Version *Full Version*: <http://arxiv.org/abs/2207.02817>

Funding This work was supported by a Dissertation Writing Fellowship awarded by the Manning College of Information and Computer Sciences, UMass Amherst to R. Addanki. In addition, this work was supported by NSF grants CCF-1934846, CCF-1908849, and CCF-1637536, awarded to A. McGregor; and NSF grants CCF-2046235, IIS-1763618, as well as Adobe and Google Research Grants, awarded to C. Musco.

Acknowledgements Most of this work was done while R. Addanki was a student at UMass Amherst. Part of this work was done while R. Addanki was a visiting student at the Simons Institute for the Theory of Computing. We thank the anonymous reviewers for their helpful suggestions.



© Raghavendra Addanki, Andrew McGregor, and Cameron Musco;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 2; pp. 2:1–2:16

Leibniz International Proceedings in Informatics



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

1 Introduction

In this work, we study sublinear query algorithms for estimating the number of edges in a simple, unweighted graph $G = (V, E)$, and for sampling uniformly random edges. Access to G is via a *Bipartite Independent Set (BIS)* oracle [10]. A query to this oracle takes as input two disjoint subsets $L, R \subseteq V$ and returns

$$\text{BIS}(L, R) = \begin{cases} '1' & \text{if there is no edge between } L \text{ and } R \\ '0' & \text{otherwise.} \end{cases}$$

Local Query Models. Prior work on sublinear query graph algorithms has largely focused on *local* queries, in particular, (i) vertex degree queries (ii) neighbor queries (output the i^{th} neighbor of a vertex) and (iii) edge existence queries [31, 33, 48]. In the literature, the first two types of queries form the *adjacency list* query model, while all three types of queries form the *adjacency matrix* query model. Under these models, a variety of graph estimation problems have been well studied, including edge counting and sampling [30, 33, 48, 51], subgraph counting [5, 17, 29], vertex cover [11, 43], and beyond [45].

For a graph with n nodes and m edges, given access only to degree queries, Feige [31] presented an algorithm for estimating m up to $(2 \pm \epsilon)$ relative error with query complexity $O(\sqrt{n} \cdot \text{poly}(1/\epsilon, \log n))$. This work also showed that any $(2 - o(1))$ -approximation algorithm requires $\Omega(n)$ queries. In the adjacency list query model, Goldreich and Ron [33] gave a $(1 \pm \epsilon)$ -approximation algorithm, with query complexity $O(n/\sqrt{m} \cdot \text{poly}(1/\epsilon, \log n))$. Recently, Eden and Rosenbaum [30] gave algorithms for near-uniform edge sampling with the same query complexity, and showed that this complexity is nearly tight.

Global Query Models. Motivated by the desire to obtain more query efficient algorithms, Beame et al. [10] studied edge estimation using *global* queries that can make use of information across the graph, including the BIS queries that we will focus on, and the related Independent Set (IS) queries, which were initially introduced in the literature on query efficient graph recovery [1, 6]. An IS query answers whether or not there exist any edges in the induced subgraph on a subset of nodes $S \subseteq V$. We refer the reader to the exposition in [10], which discusses applications of these global query models in group testing [22, 27], computational geometry [7, 18, 32], fine-grained complexity [25, 26], and decision versus counting complexity [26, 46, 49, 50].

In the IS query model, [10, 23] give a $O(\min\{\sqrt{m}, n/\sqrt{m}\} \cdot \text{poly}(\log n, 1/\epsilon))$ query algorithm for $(1 \pm \epsilon)$ approximate edge counting. In the BIS model, numerous authors [10, 26, 14] achieve $(1 \pm \epsilon)$ -approximation for edge counting and near-uniform edge sampling using just $\text{poly}(1/\epsilon, \log n)$ queries. This can be exponentially smaller than the query complexities in the IS and local queries models.

Extending the BIS query model to hypergraphs, Dell et al. [26] introduce the *coloured independence oracle* which detects the presence of a size k hyperedge. They give algorithms for hyperedge estimation and sampling using this generalized oracle. Many other variants of global queries have been studied including LINEAR, OR and CUT queries [8, 20, 47]. These queries have been applied to solving maximum matching [38, 42], minimum cut [47], triangle estimation [12, 13, 26], connectivity [8], hitting sets [15], weighted edge estimation [16], problems related to linear algebra [44], quantum algorithms [40], and full graph recovery [1, 6].

The Role of Adaptivity. Notably, for both local and global queries, most sublinear time graph algorithms are *adaptive*, i.e., a query may depend on the answers to previous queries. In many cases, it is desirable for queries to be *non-adaptive*. This allows them to be completed

independently, and might allow for the resulting algorithm to be easily implemented in massively parallel computation frameworks [37]. Non-adaptive algorithms also lead naturally to single-pass, rather than multi-pass, streaming algorithms. In fact, the BIS query model can be seen as a very restricted subset of the more general LINEAR query model, in which each query outputs the inner product of the edge indicator vector with a query vector. This model has long been studied in the graph-streaming literature [4, 39], in part due to its usefulness in giving single-pass algorithms. However, it has remained open whether non-adaptive algorithms can be given in more restricted global query models.

For these reasons, Assadi et al. [8] and Chakrabarti and Stoeckl [20] have recently sought to reduce query adaptivity under a variety of global query models, including LINEAR, OR, CUT and BIS queries. These works study the *single element recovery* problem, which is a weaker variant of uniform edge sampling, requiring that the algorithm return a single edge in G . Assadi et al. also study the problem of checking connectivity, presenting a BIS query algorithm making $\tilde{O}(n)$ queries and using three rounds of adaptivity. They give a two-round algorithm in the stronger OR query model, and show that even in this model, there is no non-adaptive algorithm for connectivity making $o(n^2)$ queries.

We note that reducing query adaptivity is also a well-studied direction in the closely related literature on group testing [28, 34]. IS and BIS oracles can be thought of as tests if there is a single element in a group of edges, where that group is required to be all edges incident on one node set (IS) or between two disjoint sets (BIS). Attempts to minimize query adaptivity have also been made for sparse recovery [35, 36, 41], submodular function maximization [9, 21], property testing [19] and multi-armed bandit learning [3].

1.1 Our Contributions

Our main result is the first *non-adaptive* algorithm for edge estimation up to $(1 \pm \epsilon)$ relative error, using $\text{poly}(1/\epsilon, \log n)$ BIS queries. Formally, we show:

► **Theorem 1** (Theorem 6 restated). *Given a graph G with n nodes and m edges, there is an algorithm that makes $O(\epsilon^{-5} \log^5 n \log^5(\log n) \log(\epsilon^{-1} \log n))$ non-adaptive BIS queries to G and returns an estimate \hat{m} satisfying: $m(1-\epsilon) \leq \hat{m} \leq m(1+\epsilon)$, with probability at least $3/5$.¹*

Prior methods for $(1 \pm \epsilon)$ error edge estimation using BIS queries are based on a binary search style approach [10, 26, 14], which is inherently adaptive, and leads to algorithms requiring $\Omega(\log^2 n)$ rounds of adaptivity. Beame et al. [10] present a non-adaptive algorithm giving a $O(\log^2 n)$ approximation factor for bipartite graphs, using $O(\log^3 n)$ queries. This algorithm can be extended to general graphs, via a simple reduction, described in Section 3.3. However, no non-adaptive results for general graphs achieving $1 \pm \epsilon$ relative error for arbitrary $\epsilon > 0$ were previously known. Even with adaptivity, the best known algorithm due to [14] has a query complexity of $O(\epsilon^{-2} \log^{11} n)$. The non-adaptive result of Theorem 1 improves upon this prior work significantly, whenever ϵ is constant with respect to n . Our second result builds on our edge estimation approach, giving the first non-adaptive BIS query algorithm that returns a near-uniformly sampled edge. Formally:

► **Theorem 2** (Theorem 7 restated). *Given a graph G with n nodes, m edges, and edge set E , there is an algorithm that makes $O(\epsilon^{-4} \log^6 n \log(\epsilon^{-1} \log n) + \epsilon^{-6} \log^5 n \log^6(\log n) \log(\epsilon^{-1} \log n))$ non-adaptive BIS queries which, with probability at least $1-\epsilon$, outputs an edge from a probability distribution P satisfying $(1-\epsilon)/m \leq P(e) \leq (1+\epsilon)/m$ for every $e \in E$.*

¹ Note that the success probability can be boosted in the standard way, by running multiple independent instantiations of the algorithm and taking their median estimate.

Prior results for near-uniform edge sampling required $\Omega(\log^3 n)$ rounds of adaptivity [14, 26]. Additionally, even ignoring adaptivity, our result improves on the best known query complexity of $O(\epsilon^{-2} \log^{14} n)$, due to [14], for large enough ϵ , including when ϵ is a constant.

By combining Theorem 2 with prior work on sublinear query graph connectivity, via edge sampling, we obtain a connectivity algorithm using two rounds for adaptivity:

► **Theorem 3** (Theorem 8 restated). *Given a graph G with n nodes, there is a 2-round adaptive algorithm that determines if G is connected with probability at least $1 - 1/n$ using $O(n \log^8 n \log(\log n))$ BIS queries.*

Theorem 3 improves on a three-round algorithm of Assadi et al. [8] and is tight in terms of adaptivity: even in the stronger OR query model (which allows checking the presence of an edge within an arbitrary subset of node pairs) no non-adaptive algorithm can make $o(n^2)$ queries. Assadi et al. gave a two-round algorithm in this stronger OR query model. Thus, Theorem 3 closes the gap between BIS queries and OR queries for the connectivity problem. We note that there is a separation from the even stronger LINEAR query model, where non-adaptive algorithms for connectivity and cut approximation are well-known [4]. Understanding if there remain natural separations between the BIS and OR query models in terms of adaptivity would be very interesting.

2 Technical Overview

In this section, we present an overview of three main results: our non-adaptive BIS query algorithm for edge estimation (Theorem 1) and near-uniform edge sampling (Theorem 2), along with our 2-round algorithm for connectivity (Theorem 3).

2.1 Edge Estimation

A simple idea to estimate the number of edges in a graph via BIS queries is to sample small random subsets of nodes and run BIS queries to check the presence of edges between them. The fraction of these queries that return ‘1’ (i.e., indicating the presence of no edge) can then be used to estimate the total number of edges in the graph. In particular, for a graph containing m edges, if the random subsets of nodes have $O(n/\sqrt{m})$ nodes in them, then we expect a ‘1’ answer with constant probability. Beame et al. [10] describe a non-adaptive algorithm along these lines, which gives a $O(\log^2 n)$ approximation using $O(\log^3 n)$ queries. Unfortunately, going beyond this coarse approximation is difficult: many dependencies due to common neighbors arise and this increases the variance of the estimators. Beame et al. handle the issue by using the coarse estimates to subdivide the graph into smaller sub-graphs, until these divided graphs only contain $\text{poly log } n$ edges, at which point all their edges can be discovered with few queries. This strategy yields a $(1 \pm \epsilon)$ approximation, however, it is inherently adaptive.

Our non-adaptive algorithm takes a different approach. Let $d(v)$ denote the degree of a node v . Suppose we could sample each node with probability $p_v \approx d(v) \cdot \epsilon^{-2}/m$ and compute the degree of the sampled nodes. Letting $\hat{m} = \sum_v \mathbb{I}[v \text{ sampled}] \cdot d(v)/p_v$, be the appropriately weighted average of the sampled degrees, it is straightforward to show that $\mathbf{E}[\hat{m}] = \sum_v d(v) = 2m$. Further, via a standard Bernstein bound, $\hat{m}/2$ gives a $(1 \pm \epsilon)$ relative error approximation to m with high probability. The challenge is showing that this type of approach can be approximated in the BIS query model.

Subsampling Nodes. The first idea, drawn from work on streaming algorithms, is to subsample the nodes of G at different rates of the form $1/\gamma^j$ where $\gamma > 1$ is constant and $j \in \{0, 1, \dots, O(\log n)\}$. At each rate, we “recover” any sampled nodes (along with a corresponding degree estimate) whose degree is roughly $d(v) \approx \epsilon^2 m / \gamma^j$. In this way, each node will be recovered with probability roughly $1/\gamma^j \approx d(v) \cdot \epsilon^{-2} / m$, as desired. We describe this subsampling procedure in Section 3.3, as part of our main algorithm EDGE-ESTIMATOR (Algorithm 3).

Recovering Heavy Nodes. The next challenge is to show that we can actually recover the appropriate nodes and degree estimates at each sampling rate. If we can approximate the degree of all nodes sampled at rate $1/\gamma^j$ up to additive error $O(\epsilon^3 \cdot m / \gamma^j)$, we will obtain a $(1 \pm \epsilon)$ relative error approximation to the degree of any node we hope to recover at that sampling rate, i.e., any node with degree roughly $\epsilon^2 m / \gamma^j$. Using these approximations, we can determine which nodes should be recovered at that rate, and form our edge estimate.

Degree Estimation via Neighborhood Size Estimation. To achieve such an additive error approximation, we use ideas from the sparse recovery and streaming literature. In particular, we implement an approach reminiscent of the Count-Min sketch algorithm [24]. The approach is described in detail in Section 3.2, where we present Algorithm ESTIMATE-DEGREE (Algorithm 2). First observe that when sampling at rate $1/\gamma^j$, conditioned on any node v being included in the sample, the expected total degree of the sampled nodes other than v is $O(m/\gamma^j)$. If we further subdivide these nodes into $\tilde{O}(1/\epsilon^3)$ random groups, the expected total degree of all nodes other than v in any group is $\tilde{O}(\epsilon^3 \cdot m/\gamma^j)$.

Now, if v is placed in group S , we can approximately upper bound its degree by the total *neighborhood size of S* . This upper bound holds approximately as long as v does not have too many neighbors in S , which it won’t with good probability. The neighborhood size of S is in turn upper bounded by the degree of v plus the total degree of other nodes in S , and thus by $d(v) + \tilde{O}(\epsilon^3 \cdot m/\gamma^j)$ in expectation. So, in expectation, this approach gives an additive $\tilde{O}(\epsilon^3 \cdot m/\gamma^j)$ error approximation to the degree of each sampled node v , with constant probability. Repeating this procedure $O(\log n)$ times, and, as in the Count-Min sketch, taking the minimum degree estimate for each node sampled at rate $1/\gamma^j$, gives us high probability approximation for such nodes.

Neighborhood Size Estimation. The final step is to implement an algorithm that can estimate the neighborhood size of the random subset of nodes S , to be used in our degree estimation procedure. We do this in Section 3.1, where we present Algorithm NEIGHBORHOOD-SIZE (Algorithm 1). This algorithm takes as input two disjoint subsets L, R and returns a $(1 \pm \epsilon)$ -approximation for the size of the neighborhood of L in R . We highlight that this may be very different than the *number of edges connecting L to R* – the neighborhood size is the number of nodes in R with at least one edge to L . This difference is critical in removing the correlations discussed previously due to common neighbors. Such correlations lead to the adaptive nature of prior algorithms [10, 26]. To estimate the size of the neighborhood of L in R , we sample the nodes in R at different rates and ask BIS queries on L and the sampled subset of R . Intuitively, when the sampling rate is the inverse of the size of the neighborhood, we will observe a ‘1’ response with constant probability. We can detect this and thus estimate the neighborhood size.

Non-adaptivity. The approach described above is inherently non-adaptive. All random sampling of nodes and random subsets can be formed ahead of time, independently of any query responses. The only catch is that to determine which nodes should be recovered at each sampling rate, i.e., those nodes with degree $d(v) \approx \epsilon^{-2} \cdot m/\gamma^j$, we need a coarse estimate to the edge count m in the first place. Fortunately, we can bootstrap such an estimate starting with a coarse $O(\log^2 n)$ -relative error approximate estimation, due to Beame et al. [10]. We then refine this estimate iteratively using Algorithm REFINE-ESTIMATE (Algorithm 4). Each refinement improves the approximation factor by ϵ , and after $O(\log_{1/\epsilon} \log n)$, refinements our estimate will result in a $(1 \pm \epsilon)$ -approximation factor. The key observation here is that each refine step does not require any additional BIS queries. Thus, our algorithm remains non-adaptive.

2.2 Uniform Edge Sampling and Connectivity

In the full version [2], we prove Theorem 2 by designing and analyzing a non-adaptive algorithm for returning a near-uniform sample among the edges of the graph. Our approach builds heavily on our edge estimation algorithm. If we knew the degree $d(v)$ of all vertices, then to sample a uniform edge, we could sample a vertex $v \in V$ with probability $d(v)/\sum_{w \in V} d(w) = \frac{d(v)}{2m}$ and return a uniform neighbor among the neighbors of v . We can observe that the probability that an edge (v, u) is sampled is $\frac{d(v)}{2m} \cdot \frac{1}{d(v)} + \frac{d(u)}{2m} \cdot \frac{1}{d(u)} = \frac{1}{m}$. I.e., this approach yields a uniformly random edge sample.

Node Sampling. We implement the above approach approximately using BIS queries. First, observe that recovered vertices in our edge estimation algorithm are sampled with probabilities roughly proportional to their degrees. We argue that we can select a random vertex from this set, which overall is equal to any vertex v with probability approximately $\frac{d(v)}{2m}$. To do so, we leverage our degree estimates, and the fact that our edge count estimator, which is the sum of scaled degrees of recovered vertices, is well-concentrated.

Random Neighbor Sampling. It remains to show how to return a uniformly random neighbor of the sampled vertex. Similar to our edge estimation procedure, we design an algorithm which takes as input two disjoint subsets L, R and returns a uniform neighbor of L in R . To do so, we demonstrate an equivalence between the substantially more powerful OR queries and BIS queries in this specific setting, and argue that an existing algorithm for OR queries can be extended to return a uniform neighbor using BIS queries. An OR query takes as input a subset of pairs of vertices and returns ‘1’ iff there is an edge in the subset queried. Building on this, in the full version [2], we present Algorithm UNIFORM-NEIGHBOR that takes as input the subset of nodes sampled at any rate $1/\gamma^j$ as in our edge estimation algorithm, and approximately returns a uniform neighbor for every vertex v sampled in this set. Similar to ESTIMATE-DEGREE (Algorithm 2), we construct $\tilde{O}(1/\epsilon^4)$ random partitions of the sampled nodes. For every vertex v in a random subset S , we return a uniform neighbor (obtained using the idea just described) of the partition of S , as the neighbor of v . If v has large degree compared to the total degree of nodes in S , which it will if it is meant to be recovered at that sampling rate, this output will most likely be a neighbor of v , and will be close to a uniformly random one.

A Two-Round Algorithm for Connectivity. Our non-adaptive edge sampling algorithm (Theorem 2) directly yields a two-round algorithm for graph connectivity (Theorem 3), improving on a prior three-round algorithm of [8]. In particular, the algorithm of [8] selects

$O(\log^2 n)$ random neighbors per vertex, and contracts the connected components of this random graph into *supernodes*. This random sampling step can be performed using one round of $\tilde{O}(n)$ BIS queries. They prove that in the contracted graph on the supernodes, there are at most $O(n \log n)$ edges. Using this fact, they then show how to identify whether all the supernodes are connected using $\tilde{O}(n)$ BIS queries and two additional rounds of adaptivity.

We follow the same basic approach: using a first round of $\tilde{O}(n)$ queries to randomly sample $O(\log^2 n)$ neighbors per vertex and contract the graph into supernodes. Once this is done, we observe that we have BIS query access to the contracted graph simply by always grouping together the set of nodes in each supernode. So, we can directly apply the non-adaptive sampling algorithm of Theorem 2 to sample edges from the contracted graph. By a coupon collecting argument, drawing $O(n \log^2 n)$ near-uniform edge samples (with replacement) from the contracted graph suffices to recover all $O(n \log n)$ edges in the graph, and thus determine connectivity of the contracted graph, and, in turn, the original graph.

2.3 Notation

Let $G(V, E)$ denote an undirected graph on vertex set V with edges $E \subseteq V \times V$. Let $|V| = n$ be the number of nodes and $|E| = m$ be the number of edges. For any set of nodes $S \subseteq V$, let $E[S] \subseteq E$ denote the edges in the induced subgraph on S . For any two disjoint sets of nodes $L, R \subseteq V$, let $E[L, R] = \{(u, v) \in E \mid u \in L, v \in R\}$ denote the edges between them. For any $v \in V$, let $\Gamma(v) = \{u \mid (u, v) \in E \text{ for some } v \in V\}$ be its set of neighbours. Let $d(v) = |\Gamma(v)|$ be its degree. For $S \subseteq V$, let $\Gamma(S) = \bigcup_{u \in S} \Gamma(u)$ and let $d(S) = \sum_{u \in S} d(u)$.

3 Non-adaptive algorithm for edge estimation

In this section, we present our non-adaptive algorithm for edge estimation using BIS queries. In Section 3.1, we describe an algorithm that takes as input two disjoint subsets L, R and returns an estimate of the size of the neighborhood $|\Gamma(L) \cap R|$. Next, in Section 3.2, we use this algorithm to give additive error approximations of degrees of all the vertices in a given subset. Finally, in Section 3.3, using the approximate degree estimates, we construct a $(1 \pm \epsilon)$ -approximate estimator for m by sampling nodes with probabilities roughly proportional to their degrees. Missing details are included in the full version [2].

3.1 Estimating the size of neighborhood

NEIGHBORHOOD-SIZE (Algorithm 1) takes as input two disjoint subsets $L, R \subseteq V$ and returns a $(1 \pm \epsilon)$ -approximation of the size of neighborhood of L in R , i.e., $|\Gamma(L) \cap R|$ using $\text{poly}(1/\epsilon, \log n)$ BIS queries. We overview the analysis of this algorithm here.

The main idea is to sample subsets of vertices in R (denoted $\hat{R}_1, \hat{R}_2, \dots$) with exponentially decreasing probability values $1/2, 1/4, 1/8, \dots$. When the sampling rate $1/2^i$ falls below $1/|\Gamma(L) \cap R|$, we expect L to no longer have any neighbors in \hat{R}_i with good probability. In particular, we can return the inverse of the smallest probability $1/2^i$ for which $\mathcal{BIS}(L, \hat{R}_i) = '1'$, as a coarse estimate for $|\Gamma(L) \cap R|$.

To boost the accuracy of this estimate, we repeat the process $T = O(\epsilon^{-2} \log(\delta^{-1} \cdot \log n))$ times, and at each sampling rate count the number of times the BIS query $\mathcal{BIS}(L, \hat{R}_i)$ returns '1'. This count is denoted $\text{count}(i)$ in Algorithm 1, and its expectation can be written in closed form as $\mathbf{E}[\text{count}(i)] = T \cdot (1 - 1/2^i)^{|\Gamma(L) \cap R|}$. Suppose $2^i \leq |\Gamma(L) \cap R| < 2^{i+1}$, then, $\mathbf{E}[\text{count}(\hat{i})] = \Theta(T)$. Via a standard Chernoff bound, it will be approximated to $(1 \pm \epsilon)$ error with high probability by $\text{count}(\hat{i})$. Thus, we can compute an accurate estimate of the

neighborhood size by inverting our estimate of $\mathbf{E}[\text{count}(\hat{i})]$, as $\log_{(1-1/2^{\hat{i}})}(\text{count}(\hat{i})/T)$. We identify the appropriate \hat{i} in line 12 of Algorithm 1, and compute the corresponding estimate in lines 13-14. There is one edge case handled in line 13: if $|\Gamma(L) \cap R| = 1$ we will have $\hat{i} = 0$, and $\text{count}(\hat{i}) = 0$. The final error bound for Algorithm 1 is stated below.

► **Lemma 4.** *Algorithm 1 uses $O(\epsilon^{-2} \log n \log(\delta^{-1} \cdot \log n))$ BIS queries and returns an estimate $\eta_{\text{est}}(L)$ of $|\Gamma(L) \cap R|$ such that with probability at least $1 - \delta$,*

$$(1 - \epsilon) \cdot |\Gamma(L) \cap R| \leq \eta_{\text{est}}(L) \leq (1 + \epsilon) \cdot |\Gamma(L) \cap R|.$$

■ **Algorithm 1** NEIGHBORHOOD-SIZE: Estimating the neighborhood size of L in R .

Input: $L, R \subseteq V$, approximation error ϵ , failure probability δ .
Output: $\eta_{\text{est}}(L)$ as an estimate of $|\Gamma(L) \cap R|$.

- 1: Initialize $\eta_{\text{est}}(L) \leftarrow 0$.
- 2: **for** $i = 0, 1, \dots, \log_2 n$ **do**
- 3: $\text{count}(i) \leftarrow 0$.
- 4: **for** $t = 1, 2, \dots, T = 2e^8 \ln(\log n / \delta) \cdot \epsilon^{-2}$ **do**
- 5: $\hat{R}_i^t \leftarrow \{u \in R \mid u \text{ is included independently with probability } 1/2^i\}$.
- 6: $\text{count}(i) = \text{count}(i) + \text{BIS}(L, \hat{R}_i^t)$
- 7: **end for**
- 8: **end for**
- 9: **if** $\text{count}(0) = T$ **then**
- 10: **return** $\eta_{\text{est}}(L) = 0$.
- 11: **else**
- 12: Set $\hat{i} \leftarrow \max \{i \mid \frac{\text{count}(i)}{T} < \frac{(1-\epsilon)}{2e^2}\}$.
- 13: **if** $\hat{i} = 0$ **then return** $\eta_{\text{est}}(L) = 1$.
- 14: **else return** $\eta_{\text{est}}(L) = \log_{(1-1/2^{\hat{i}})}(\text{count}(\hat{i})/T)$.
- 15: **end if**
- 16: **end if**

3.2 Finding good approximation for degrees of vertices

We now describe how to use NEIGHBORHOOD-SIZE (Algorithm 1) to estimate the degrees of all vertices in a given subset $S \subseteq V$ up to additive error depending on the total degree of S . Our approach is inspired by the Count-Min sketch algorithm [24]. We randomly partition S into subsets $S^1, S^2, \dots, S^\lambda$ where $\lambda = O(\epsilon^{-3} \log^2 n)$. The choice of the parameter λ is based on the analysis in Section 3.3. For each S^i , we estimate the size of the neighborhood of S^i in $V \setminus S^i$ using NEIGHBORHOOD-SIZE. We then return this neighborhood size estimate as the degree estimate for all vertices in S^i . For $v \in S^i$, $|\Gamma(S^i) \cap V \setminus S^i|$ is nearly an overestimate for $d(v)$, as long as v has few neighbors in S^i , which it will with high probability. Additionally, it is not too large an overestimate – we can observe that $|\Gamma(S^i) \cap V \setminus S^i| - d(v) \leq d(S^i \setminus v)$. I.e., the error in the overestimate is at most the total degree of the other nodes in S^i . In expectation, this error is at most $\frac{d(S)}{\lambda} = O\left(d(S) \cdot \frac{\epsilon^3}{\log^2 n}\right)$ due to our random choice of S^i .

As in the Count-Min sketch algorithm, to obtain high probability estimates, we repeat the process $T = O(\log n)$ times and assign the minimum among the neighborhood estimates as the degree estimate of $d(v)$. The full approach is given in Algorithm 2 (ESTIMATE-DEGREE) and the error bound in Lemma 5 below. We set the failure probability, $\delta = O(\epsilon^3 / \log^4 n)$, for each of the calls to NEIGHBORHOOD-SIZE, to ensure that the total failure probability of Algorithm 2 is at most $O(1/\log n)$. As we make at most $T \cdot \lambda$ calls to the Algorithm NEIGHBORHOOD-SIZE, the total BIS queries used is $O(\log n \cdot \epsilon^{-3} \log^2 n \cdot \epsilon^{-2} \log n \log(\log^4 n \cdot \log n)) = O(\epsilon^{-5} \log^4 n \log(\log n))$. Formally, we have:

► **Lemma 5.** For any $S \subseteq V$, Algorithm 2 uses $O(\epsilon^{-5} \log^4 n \log(\log n))$ BIS queries and with probability $1 - O(1/\log n)$, returns degree estimates $\widehat{d}(v)$ for every $v \in S$ satisfying:

$$d(v)(1 - \epsilon) \leq \widehat{d}(v) \leq d(v) + \frac{\epsilon^3}{\log^2 n} \cdot d(S).$$

■ **Algorithm 2** ESTIMATE-DEGREE: Obtain additive approximate degree estimates.

Input: S is a subset of V , ϵ is approximation error.

Output: Degree estimates of vertices in S .

```

1: Initialize  $\widehat{d}(v) \leftarrow n$  for every  $v \in S$ .
2: for  $t$  in  $\{1, 2, \dots, O(\log n)\}$  do
3:   Form a random partition of  $S$  into  $S^{t,1}, S^{t,2}, \dots, S^{t,\lambda}$  where  $\lambda = O(\epsilon^{-3} \log^2 n)$ .
4:   for every set  $S^{t,a}$  where  $a \in [\lambda]$  do
5:      $\eta_{\text{est}}(S^{t,a}) \leftarrow \text{NEIGHBORHOOD-SIZE}(S^{t,a}, V \setminus S^{t,a}, \epsilon/3, \delta)$ , where  $\delta = O(\epsilon^3 / \log^4 n)$ .
6:     For all  $v \in S^{t,a}$ , set  $\widehat{d}(v) \leftarrow \min\{\widehat{d}(v), \eta_{\text{est}}(S^{t,a})\}$ .
7:   end for
8: end for
9: return  $\widehat{d}(v)$  for every  $v \in S$ .

```

3.3 Edge Estimation

In this section, we describe the algorithm EDGE-ESTIMATOR (Algorithm 3) that obtains a $(1 \pm \epsilon)$ -approximation for the number of edges m . Missing details are presented in the full version [2].

Our Approach. A naive strategy to estimate the number of edges m is to sample $\widetilde{O}(\epsilon^{-2})$ nodes uniformly, and estimate m as $n/2$ times the average degree of the sampled nodes. However, the variance of such an estimator depends on the maximum degree, which could be as high as n . To fix this issue, we would like to sample vertices with probabilities proportional to their degrees. In particular, we sample vertices at different rates $1/\gamma^j$, where $\gamma > 1$ is a constant and $j \in \{0, 1, \dots, \log n\}$. We use the term j^{th} level to refer to the sampling rate $1/\gamma^j$. Our estimator is given by: $\widehat{m} = \sum_v \mathbb{I}[v \text{ sampled}] \cdot d(v)/p_v$, be the appropriately weighted average of the sampled degrees. It is straightforward to show that $\mathbf{E}[\widehat{m}] = \sum_v d(v) = 2m$ and as argued in section 1, it is also concentrated around $2m$ with high probability. It is easy to observe that when a vertex v is sampled at rate $\widetilde{O}(\epsilon^{-2}d(v)/m)$, its contribution to \widehat{m} is $\widetilde{O}(\epsilon^2m)$. In other words, we need to detect the event that $d(v) \approx \epsilon^2m/\gamma^j$, for some sampling level j . If we identify $\widetilde{O}(\epsilon^{-2})$ such vertices, \widehat{m} will be an accurate estimate of the total edges, after appropriate scaling. However, there are three main challenges in implementing this approach which we detail below.

Approximate degrees. Algorithm ESTIMATE-DEGREE returns degree estimates with an additive approximation error of $O(\epsilon^3 \log^{-2} n \cdot d(S_j))$ at sampling level j . $\widetilde{O}(\epsilon^3m/\gamma^j)$ at sampling level j . It is easy to see that $\mathbf{E}[d(S_j)] = O(m/\gamma^j)$. From Markov's inequality and union bound, we have that: $d(S_j) = O(m \log n / \gamma^j)$ for all $j \in [L]$ with probability at least $3/4$. Therefore, the additive approximation error term is $\widetilde{O}(\epsilon^3 \cdot m/\gamma^j)$. To include the contribution of a vertex v in the estimator, we must ensure that this error term is small in a relative sense – i.e., at most $O(\epsilon \cdot d(v))$. This holds whenever $d(v) = \widetilde{\Omega}(\epsilon^2m/\gamma^j)$. Observe that this corresponds to the threshold we mentioned earlier. Therefore, our goal is to identify all vertices at every level j that pass the threshold of $\widetilde{\Omega}(\epsilon^2m/\gamma^j)$. When that happens, we say that the vertex v has been recovered at level j and can be safely included in our estimator.

Knowledge of m . As we do not know the value of m , we start with an $O(\log^2 n)$ -relative error approximate estimate, obtained using the Algorithm COARSEESTIMATOR in Beame et al. [10], as follows: Given a partition $L, R \subseteq V$, Algorithm COARSEESTIMATOR returns an approximate estimate for the number of edges between L and R , given by: $\frac{m(L,R)}{8 \log n} \leq \bar{m}(L,R) \leq 8 \log n \cdot m(L,R)$, where $m(L,R) = |E[L,R]|$. However, using a simple reduction, we can convert this estimate into an $O(\log^2 n)$ -relative error approximation for total edges in the graph G . First, we partition the graph uniformly into two sets of vertices L and R . We set our estimate for edges as: $\bar{m} = O(\log n) \cdot \bar{m}(L,R)$. From Lemma 3.2 in [10], $m(L,R) = \Theta(m)$ when $m \geq 2$, with constant probability. So, our initial estimate for refinement, \bar{m} , satisfies: $m \leq \bar{m} \leq O(\log^2 n) \cdot m$.

We repeatedly refine the approximate estimate using Algorithm REFINE-ESTIMATE, until we get a $(1 \pm \epsilon)$ -relative error approximation of m . Each *refinement* improves the approximation factor from the previous stage by a multiplicative factor of ϵ . We note that each refinement does not require any additional BIS queries and uses the available approximate degree estimates.

Boundary Vertices. We partition the space of possible degrees, i.e., $[0, n]$ into geometrically decreasing partitions, called *levels*. Each level is represented by an integer in $[0, L]$. We recover a vertex at a particular level $j \in [0, L]$, if it is sampled with probability corresponding to the level, i.e., γ^{-j} and it passes the threshold mentioned earlier. It is possible that some vertices have degrees close to the threshold values at each sampling level. We denote the set of such vertices as $V_{\text{boundary}} \subseteq V$. For such boundary vertices, as we use approximate degree estimates, they might be recovered at a level different from their true levels (defined with respect to exact degrees). Such a scenario could potentially affect the contribution of the recovered vertex in our estimator by an additional multiplicative factor dependent on γ and the difference between recovered level and true level. As a result, our estimator might not be a $(1 \pm \epsilon)$ -relative error approximation anymore. We get around this limitation by dividing the region between any two consecutive levels into $B = O(1/\epsilon)$ buckets, where ϵ denotes the approximation parameter, and shifting the boundaries of all the levels by a random shift selected uniformly from the first B buckets. We account for this by changing the sampling rates to $\gamma^{-\mu(j)}$ where $\mu(j)$ encodes the random shift. We set the parameter corresponding to the sampling probability, $\gamma = 1/(1 - \epsilon)$.

With the random shift of the level boundaries, we ensure that every vertex will lie close to the boundary with probability at most ϵ . Moreover, we argue that every boundary vertex is recovered at its true level or level adjacent to its true level. Therefore, the total contribution of V_{boundary} to our edge estimator is $O(\epsilon m)$.

Random Boundary Shift. The region between two consecutive levels is divided into B buckets with the boundaries of buckets proportional to the values given by: $\{[1/\gamma^B, 1/\gamma^{B-1}), \dots, [1/\gamma^2, 1/\gamma), [1/\gamma, 1)\}$. We select a random integer offset for shifting our levels, denoted by s , which is selected uniformly at random from $[0, B)$. Now, the level boundaries are located at values proportional to $\gamma^{-\mu(j)}$ where $\mu(j) = j \cdot B - s$ and $0 \leq j \leq L$. Observe that the number of sampling levels is given by $L = \frac{2}{B} \cdot \log_\gamma n + 1 \leq \log n + 1$. Combining everything, the exact level boundaries are dependent on the estimate \bar{m} and given by $O\left(\frac{\bar{m}}{\gamma^{\mu(j)}} \cdot \frac{\epsilon^2}{\log n}\right)$, for every $j \in [0, L]$.

3.3.1 Overview of Algorithm Edge-Estimator

In Algorithm EDGE-ESTIMATOR, we construct sets $V = S_0 \supseteq S_1 \supseteq \dots \supseteq S_L$ where a set S_j (for all $j \geq 2$) is obtained by sampling vertices in S_{j-1} with probability $1/\gamma^B$. The set S_1 is obtained by sampling vertices in V with probability $1/\gamma^{-s+B}$. Our sampling scheme results in each vertex being included in a set S_j with probability $1/\gamma^{\mu(j)}$. As described previously, with constant probability, $d(S_j) = O(m \log n / \gamma^{\mu(j)})$, for all j . Using Algorithm 2, we obtain approximate degree estimates of vertices in S_j for every sampling level $j \leq L$ with an approximation error of $O(\epsilon^3 / \log^2 n \cdot d(S_j)) = O(m \epsilon^3 / \gamma^{\mu(j)} \log n)$. In order that this error is small, we need to recover vertices $v \in S_j$ with high degree, such that $d(v) = \tilde{\Omega}(\frac{m}{\gamma^{\mu(j)}} \cdot \frac{\epsilon^2}{\log n})$. As we do not know m , we bootstrap it with a $O(\log^2 n)$ -relative error approximate estimate due to [10], denoted by \bar{m}_0 . We repeatedly refine the estimate $T = 2 \log_{1/\epsilon} \log n$ times, using REFINE-ESTIMATE (Algorithm 4), where the estimate \bar{m}_{t-1} is used to construct an improved estimate \bar{m}_t . We return the estimate \bar{m}_T as our final estimate for m . The constants used c_1, c_2 satisfy $c_1 \leq c_2/10$ and $c_2 \geq 50$.

■ **Algorithm 3** EDGE-ESTIMATOR: Non-adaptive algorithm for estimating edges.

Input: V set of n vertices and $\epsilon > 0$ error parameter.

Output: Estimate \hat{m} of number of edges in G .

- 1: Scale $\epsilon \leftarrow \frac{\epsilon}{600 \log_{1/\epsilon} \log n}$ and initialize $\gamma \leftarrow 1/(1 - \epsilon)$ and $B \leftarrow 2/\epsilon$.
- 2: Let s be an integer selected uniformly at random from the interval $[0, B)$.
- 3: Let $\mu(j) \leftarrow -s + j \cdot B$ for every integer j in the interval $[0, \frac{2}{B} \cdot \log_\gamma n + 1]$.
- 4: Initialize $S_0 \leftarrow V$ and construct S_1 by sampling vertices in S_0 with probability $1/\gamma^{\mu(1)}$.
- 5: Construct $S_2 \supseteq \dots \supseteq S_L$ for $L = \frac{2}{B} \cdot \log_\gamma n$ where each S_j is obtained by sampling vertices in $S_{j-1} \forall j \geq 2$, independently with probability $1/\gamma^B$.
- 6: **for** $j = 0, 1, \dots, L$ **do**
- 7: Run ESTIMATE-DEGREE (S_j) to obtain the estimates $\hat{d}_j(v)$ for all $v \in S_j$ satisfying:

$$(1 - \epsilon)d(v) \leq \hat{d}_j(v) \leq d(v) + \frac{c_1 \epsilon^3 \cdot m}{\log n \cdot \gamma^{\mu(j)}}.$$
- 8: **end for**
- 9: Construct a random partition L, R of V . Let \bar{m}_0 be the $O(\log n)$ -approximate estimate from the Algorithm COARSEESTIMATOR in Beame et al. [10] on the partition L, R .
- 10: Set $\bar{m}_0 \leftarrow \max\{2, O(\log n) \cdot \bar{m}_0\}$, so that we have $m \leq \bar{m}_0 \leq O(\log^2 n) \cdot m$.
- 11: **for** $t = 1, 2, \dots, T = 2 \log_{1/\epsilon} \log n$ **do**
- 12: \bar{m}_t is assigned the output of REFINE-ESTIMATE that takes as input approximate degree values $\hat{d}_j(v) \forall v \in S_j \forall j \in [L]$, the previous estimate \bar{m}_{t-1} and the iteration t .
- 13: **end for**
- 14: **return** $\hat{m} \leftarrow \bar{m}_T$.

3.3.2 Overview of Algorithm REFINE-ESTIMATE

Suppose we are given an initial estimate \bar{m} satisfying $m \leq \bar{m} \leq (1 + \alpha)m$ for some unknown approximation factor α satisfying $\epsilon \leq \alpha \leq O(\log^2 n)$. We set the threshold value for recovering a vertex at a level j as $\frac{\bar{m}}{\gamma^{\mu(j)}} \cdot \frac{c_2 \epsilon^2}{\log n}$ where c_2 is a constant. So, when a vertex v , that hasn't been recovered at a smaller level yet, with degree estimate $\hat{d}_j(v)$ (obtained from Algorithm 3) satisfies $\hat{d}_j(v) \geq \frac{\bar{m}}{\gamma^{\mu(j)}} \cdot \frac{c_2 \epsilon^2}{\log n}$, we set the level of recovery $\hat{\ell}(v) = j$ and recovered flag $r(v) = 1$.

2:12 Non-Adaptive Edge Counting and Sampling via BIS Queries

From construction, we can observe that once a vertex is recovered at a particular level it is not available to be recovered at higher level later. Our estimator is the summation of terms $\gamma^{\mu(\hat{\ell}(v))} \cdot \hat{d}(v)$ for every v satisfying $r(v) = 1$. We normalize \hat{m} by adding an additional term of $(\epsilon \log \log n)^t \bar{m}_0$ in iteration t , to ensure that after every call to REFINE-ESTIMATE (Algorithm 4), the estimate returned \hat{m} , satisfies: $m \leq \hat{m} \leq m(1 + \epsilon \cdot \log \log n \cdot \alpha)$ (see the full version for additional details [2]). After $T = 2 \log_{1/\epsilon} \log n$ iterations of REFINE-ESTIMATE, we return the estimate \bar{m}_T as our final estimate in EDGE-ESTIMATOR (Algorithm 3).

■ **Algorithm 4** REFINE-ESTIMATE: Refines the current estimate of number of edges.

Input: \bar{m} satisfying $m \leq \bar{m} \leq m(1+\alpha)$, approximate degree values $\hat{d}_j(v) \forall v \in S_j \forall j \in [L]$ obtained using Algorithm 3, \bar{m}_0 , and iteration t .

Output: Estimate \hat{m} satisfying $m \leq \hat{m} \leq m(1 + \epsilon \cdot \alpha \cdot \log \log n)$ of number of edges in G .

- 1: Initialize $\hat{m} \leftarrow 0$.
- 2: Initialize $r(v) \leftarrow 0$ for all v (indicator if v has been recovered yet).
- 3: **for** $j = 0, 1, \dots, L$ **do**
- 4: **for** $v \in S_j$ **do**
- 5: **if** $r(v) = 0$ and $\hat{d}_j(v) \geq \frac{\bar{m}}{\gamma^{\mu(j)}} \cdot \frac{c_2 \epsilon^2}{\log n}$ **then**
- 6: $\hat{m} \leftarrow \hat{m} + \gamma^{\mu(j)} \cdot \hat{d}_j(v)$.
- 7: $\hat{\ell}(v) \leftarrow j$ and $r(v) \leftarrow 1$. ▷ Used in the analysis.
- 8: **end if**
- 9: **end for**
- 10: **end for**
- 11: **if** $t < T = 2 \log_{1/\epsilon} \log n$ **then**
- 12: $\hat{m} = \hat{m}/2 + (\epsilon \log \log n)^t \bar{m}_0$. ▷ We normalize \hat{m} so that we have $\hat{m} \geq m$.
- 13: **else**
- 14: $\hat{m} = \hat{m}/2$.
- 15: **end if**
- 16: **return** \hat{m} .

3.3.3 Final Guarantees of EDGE-ESTIMATOR

Using Bernstein's inequality, we argue that in iteration t , we can improve the approximation factor of the previous estimate \bar{m}_{t-1} by a multiplicative factor of ϵ in the new estimate \bar{m}_t . After $T = O(\log_{1/\epsilon} \log n)$ iterations, the edge estimate will be a $(1 \pm \epsilon)$ -relative error approximation satisfying:

► **Theorem 6.** *Given a graph G with n nodes and m edges, there is an algorithm that makes $O(\epsilon^{-5} \log^5 n \log^5(\log n) \log(\epsilon^{-1} \log n))$ non-adaptive BIS queries to G and returns an estimate \hat{m} satisfying: $m(1 - \epsilon) \leq \hat{m} \leq m(1 + \epsilon)$, with probability at least $3/5$.*

4 Uniform Edge Sampling

In this section, we give brief overview of an algorithm that returns a near-uniformly sampled edge from the graph using $\text{poly}(\log n, 1/\epsilon)$ BIS queries. We present the complete details in the full version [2]. Our algorithm extends EDGE-ESTIMATOR (Alg. 3) and is based on the following idea. Suppose we know the degrees of all the vertices. In order to sample a uniform edge, we can sample a vertex v with probability $d(v)/\sum_{w \in V} d(w) = d(v)/2m$ and return a uniform neighbor among the neighbors of v . The probability that an edge $e = (v, u)$ is sampled is $d(v)/2m \cdot 1/d(v) + d(u)/2m \cdot 1/d(u) = 1/m$.

In order to return a uniform edge, using the above approach, to our setting, there are two challenges. First, we do not know the degrees (or approximate degrees) of *all the vertices*. This is because the set of recovered vertices in REFINE-ESTIMATE (Alg. 4) at a particular level j , i.e., $r(v) = 1$, is a subset of the sampled vertices S_j . Only the recovered vertices at any particular level have accurate degree estimates, i.e., $\widehat{d}_j(v) \approx (1 \pm \epsilon)d(v)$. Secondly, vertices are recovered at different levels and are therefore sampled with different probabilities. In order to return a uniform edge using the previously discussed idea, we must return a single vertex among the set of recovered vertices with probability dependent on the sampling probability at which the vertex was recovered, and its approximate degree.

We address these two challenges, by suitably modifying EDGE-ESTIMATOR (Alg. 3) and returning a vertex v , among the recovered vertices, with probability proportional to $\gamma^{\widehat{\ell}(v)} \cdot \widehat{d}_{\widehat{\ell}(v)}(v)$ where $\widehat{\ell}(v)$ is the level at which it is recovered, and $\widehat{d}_{\widehat{\ell}(v)}(v)$ is the degree estimate at the level of recovery. From Section 3.3, we know that our estimator: $\sum_{v \text{ is recovered}} \gamma^{\widehat{\ell}(v)} \cdot \widehat{d}(v)$, is concentrated around $2m$ (Theorem 6). Combining all the above, we obtain the following result about sampling an edge:

► **Theorem 7.** *Given a graph G with n nodes, m edges, and edge set E , there is an algorithm that makes $O(\epsilon^{-4} \log^6 n \log(\epsilon^{-1} \log n) + \epsilon^{-6} \log^5 n \log^6(\log n) \log(\epsilon^{-1} \log n))$ non-adaptive BIS queries which, with probability at least $1 - \epsilon$, outputs an edge from a probability distribution P satisfying $(1 - \epsilon)/m \leq P(e) \leq (1 + \epsilon)/m$ for every $e \in E$.*

Graph Connectivity. Using the non-adaptive uniform sampling algorithm, we obtain a 2-round adaptive algorithm for determining graph connectivity, by building upon the work of [8]. See Section 2.2 for an outline of this result. Details are deferred to the full version [2].

► **Theorem 8.** *Given a graph G with n nodes, there is a 2-round adaptive algorithm that determines if G is connected with probability at least $1 - 1/n$ using $O(n \log^8 n \log \log n)$ BIS queries.*

5 Conclusion and Open Questions

In this paper, we presented the first $(1 \pm \epsilon)$ relative error non-adaptive algorithms for edge estimation and sampling using BIS queries. It would be interesting to investigate if better dependencies on ϵ than given by our algorithms can be obtained. Further, using Independent Set (IS) queries, *adaptive* algorithms for edge estimation with optimal query complexity $O(\min\{\sqrt{m}, n/\sqrt{m}\} \cdot \text{poly}(\log n, 1/\epsilon))$ were obtained only recently [10, 23]. It would be interesting to see if we can extend our techniques to study *non-adaptive* algorithms for edge estimation using IS queries or in the standard adjacency list query model, studied in the sublinear time graph algorithms literature.

References

- 1 Hasan Abasi and Bshouty Nader. On learning graphs with edge-detecting queries. In *Algorithmic Learning Theory (ALT)*, pages 3–30, 2019.
- 2 Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. *arXiv*, 2022. [arXiv:2207.02817](https://arxiv.org/abs/2207.02817).
- 3 Arpit Agarwal, Shivani Agarwal, Sepehr Assadi, and Sanjeev Khanna. Learning with limited rounds of adaptivity: Coin tossing, multi-armed bandits, and ranking from pairwise comparisons. In *Proceedings of the 30th Annual Conference on Computational Learning Theory (COLT)*, pages 39–75, 2017.

- 4 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467, 2012.
- 5 Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. Sublinear-time algorithms for counting star subgraphs via edge sampling. *Algorithmica*, 80(2):668–697, 2018.
- 6 Dana Angluin and Jiang Chen. Learning a hidden graph using $o(\log n)$ queries per edge. *Journal of Computer and System Sciences*, 74(4):546–556, 2008.
- 7 Boris Aronov and Sarel Har-Peled. On approximating the depth and related problems. *SIAM Journal on Computing*, 38(3):899–921, 2008.
- 8 Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and OR queries. In *29th Annual European Symposium on Algorithms, (ESA)*, pages 7:1–7:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 9 Eric Balkanski and Yaron Singer. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1138–1151, 2018.
- 10 Paul Beame, Sarel Har-Peled, Sivaramkrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Transactions on Algorithms (TALG)*, 16(4):1–27, 2020.
- 11 Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *Proceedings of the 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 873–884, 2022.
- 12 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Hyperedge estimation using polylogarithmic subset queries. *arXiv*, 2019. [arXiv:1908.04196](https://arxiv.org/abs/1908.04196).
- 13 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. On triangle estimation using tripartite independent set queries. *Theory of Computing Systems*, pages 1–28, 2021.
- 14 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Faster counting and sampling algorithms using colorful decision oracle. In *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2022.
- 15 Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, Gopinath Mishra, and Saket Saurabh. Parameterized query complexity of hitting set using stability of sunflowers. In *29th International Symposium on Algorithms and Computation*, 2018.
- 16 Arijit Bishnu, Arijit Ghosh, Gopinath Mishra, and Manaswi Paraashar. Efficiently sampling and estimating from substructures using linear algebraic queries. *arXiv*, 2019. [arXiv:1906.07398](https://arxiv.org/abs/1906.07398).
- 17 Amartya Shankha Biswas, Talya Eden, and Ronitt Rubinfeld. Towards a decomposition-optimal algorithm for counting and sampling arbitrary motifs in sublinear time. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, 2021.
- 18 Sergio Cabello and Miha Ježič. Shortest paths in intersection graphs of unit disks. *Computational Geometry*, 48(4):360–367, 2015.
- 19 Clément L Canonne and Tom Gur. An adaptivity hierarchy theorem for property testing. *Computational Complexity*, 27(4):671–716, 2018.
- 20 Amit Chakrabarti and Manuel Stoekli. The element extraction problem and the cost of determinism and limited adaptivity in linear queries. *arXiv*, 2021. [arXiv:2107.05810](https://arxiv.org/abs/2107.05810).
- 21 Chandra Chekuri and Kent Quanrud. Parallelizing greedy for submodular set function maximization in matroids and beyond. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 78–89, 2019.
- 22 Chao L Chen and William H Swallow. Using group testing to estimate a proportion, and to test the binomial model. *Biometrics*, pages 1035–1046, 1990.
- 23 Xi Chen, Amit Levi, and Erik Waingarten. Nearly optimal edge estimation with independent set queries. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2916–2935, 2020.

- 24 Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
- 25 Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. *ACM Transactions on Computation Theory (TOCT)*, 13(2):1–24, 2021.
- 26 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colourful decision oracle. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2201–2211, 2020.
- 27 Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.
- 28 Dingzhu Du, Frank K Hwang, and Frank Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 2000.
- 29 Talya Eden, Dana Ron, and C Seshadhri. On approximating the number of k -cliques in sublinear time. *SIAM Journal on Computing*, 49(4):747–771, 2020.
- 30 Talya Eden and Will Rosenbaum. On sampling edges almost uniformly. In *1st Symposium on Simplicity in Algorithms (SOSA)*, 2018.
- 31 Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4):964–984, 2006.
- 32 Aleksei V Fishkin. Disk graphs: A short survey. In *International Workshop on Approximation and Online Algorithms*, pages 260–264. Springer, 2003.
- 33 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008.
- 34 Piotr Indyk, Hung Q Ngo, and Atri Rudra. Efficiently decodable non-adaptive group testing. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1126–1142, 2010.
- 35 Piotr Indyk, Eric Price, and David P Woodruff. On the power of adaptivity in sparse recovery. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 285–294, 2011.
- 36 Akshay Kamath and Eric Price. Adaptive sparse recovery with limited adaptivity. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2729–2744, 2019.
- 37 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010.
- 38 Lidiya Khalidah binti Khalil and Christian Konrad. Constructing large matchings via query access to a maximal matching oracle. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2020.
- 39 Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.
- 40 Ashley Montanaro and Changpeng Shao. Quantum algorithms for learning a hidden graph. In *17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.
- 41 Vasileios Nakos, Xiaofei Shi, David P Woodruff, and Hongyang Zhang. Improved algorithms for adaptive compressed sensing. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*, 2018.
- 42 Noam Nisan. The demand query model for bipartite matching. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 592–599, 2021.
- 43 Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1123–1131, 2012.
- 44 Cyrus Rashtchian, David P Woodruff, and Hanlin Zhu. Vector-matrix-vector queries for solving linear algebra, statistics, and graph problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, 2020.

2:16 Non-Adaptive Edge Counting and Sampling via BIS Queries

- 45 Dana Ron. Sublinear-time algorithms for approximating graph parameters. In *Computing and Software Science*, pages 105–122. Springer, 2019.
- 46 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):1–19, 2016.
- 47 Aviad Rubinfeld, Tselil Schramm, and S. Matthew Weinberg. Computing Exact Minimum Cuts Without Knowing the Graph. In *Proceedings of the 9th Conference on Innovations in Theoretical Computer Science (ITCS)*, 2018.
- 48 C Seshadhri. A simpler sublinear algorithm for approximating the triangle count. *arXiv*, 2015. [arXiv:1505.01927](https://arxiv.org/abs/1505.01927).
- 49 Larry Stockmeyer. The complexity of approximate counting. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 118–126, 1983.
- 50 Larry Stockmeyer. On approximation algorithms for $\#$ p. *SIAM Journal on Computing*, 14(4):849–861, 1985.
- 51 Jakub Tětek and Mikkel Thorup. Edge sampling and graph parameter estimation via vertex neighborhood accesses. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1116–1129, 2022.

Hardness of Token Swapping on Trees

Oswin Aichholzer ✉

Technische Universität Graz, Austria

Erik D. Demaine ✉ 

CSAIL, Massachusetts Institute of Technology, Cambridge, MA, USA

Matias Korman ✉

Siemens Electronic Design Automation, Wilsonville, OR, USA

Anna Lubiw ✉ 

Cheriton School of Computer Science, University of Waterloo, Canada

Jayson Lynch ✉

Cheriton School of Computer Science, University of Waterloo, Canada

Zuzana Masárová ✉

IST Austria, Klosterneuburg, Austria

Mikhail Rudoy ✉

LeapYear Technologies, San Francisco, CA, USA

Virginia Vassilevska Williams ✉

CSAIL, Massachusetts Institute of Technology, Cambridge, MA, USA

Nicole Wein ✉

DIMACS, Rutgers University, Piscataway, NJ, USA

Abstract

Given a graph where every vertex has exactly one labeled token, how can we most quickly execute a given permutation on the tokens? In *(sequential) token swapping*, the goal is to use the shortest possible sequence of *swaps*, each of which exchanges the tokens at the two endpoints of an edge of the graph. In *parallel token swapping*, the goal is to use the fewest *rounds*, each of which consists of one or more swaps on the edges of a matching. We prove that both of these problems remain NP-hard when the graph is restricted to be a tree.

These token swapping problems have been studied by disparate groups of researchers in discrete mathematics, theoretical computer science, robot motion planning, game theory, and engineering. Previous work establishes NP-completeness on general graphs (for both problems), constant-factor approximation algorithms, and some poly-time exact algorithms for simple graph classes such as cliques, stars, paths, and cycles. Sequential and parallel token swapping on *trees* were first studied over thirty years ago (as “sorting with a transposition tree”) and over twenty-five years ago (as “routing permutations via matchings”), yet their complexities were previously unknown.

We also show limitations on approximation of sequential token swapping on trees: we identify a broad class of algorithms that encompass all three known polynomial-time algorithms that achieve the best known approximation factor (which is 2) and show that no such algorithm can achieve an approximation factor less than 2.

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

Keywords and phrases Sorting, Token swapping, Trees, NP-hard, Approximation

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.3

Related Version *Full Version*: <https://arxiv.org/abs/2103.06707>

Funding *Anna Lubiw*: Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Jayson Lynch: Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).



© Oswin Aichholzer, Erik D. Demaine, Matias Korman, Anna Lubiw, Jayson Lynch, Zuzana Masárová, Mikhail Rudoy, Virginia Vassilevska Williams, and Nicole Wein; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 3; pp. 3:1–3:15



Leibniz International Proceedings in Informatics

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

Zuzana Masárová: Supported by Wittgenstein Prize, Austrian Science Fund (FWF), grant no. Z 342-N31.

Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.

Nicole Wein: Supported by a grant to DIMACS from the Simons Foundation (820931). This work was done while the author was at MIT.

Acknowledgements This research was initiated at the 34th Bellairs Winter Workshop on Computational Geometry, co-organized by Erik Demaine and Godfried Toussaint, held on March 22–29, 2019 in Holetown, Barbados. We thank the other participants of that workshop for providing a stimulating research environment.

1 Introduction

Imagine n distinctly labeled tokens placed without collisions on the n vertices of a graph G . For example, these n tokens might represent (densely packed) movable *agents* – robots, people, packages, shipping containers, data packets, etc. – while the n vertices represent possible agent locations. Now suppose we want to move the tokens/agents around, for example, to bring certain shipping containers to the loading side of a cargo ship. In particular, we can suppose every token has a given start vertex and destination vertex, and the goal is to move every token to its desired destination. Because every vertex has a token (agents are densely packed), a natural reconfiguration operation is to *swap* two adjacent tokens/agents, that is, to exchange the tokens on the two endpoints of a given edge in G . In this paper, we study token reconfiguration by swaps from a given start configuration to a given destination configuration with two natural objective functions:

1. **(Sequential) Token Swapping** (a.k.a. “sorting with a transposition graph” [2]): Minimize the number of swaps, i.e., the total work required to reconfigure.
2. **Parallel Token Swapping** (a.k.a. “routing permutations via matchings” [3]): Minimize the number of rounds of simultaneous swaps (where the edges defining the swaps form a matching, so avoid conflicting shared endpoints), i.e., the total execution time or makespan required to reconfigure.

These reconfiguration problems can be cast in terms of the symmetric group. Each possible reconfiguration step – swapping along one edge in the sequential problem, or swapping along every edge of a matching in the parallel problem – is a particular permutation on the n tokens (an element of the symmetric group S_n). Assuming the graph is connected, these permutations generate S_n , defining a *Cayley graph* C [9] where each node π in C corresponds to a permutation π of the tokens (a collision-free placement of the tokens) and an undirected edge connects two nodes π_1, π_2 in C if there is a reconfiguration step (swapping an edge or matching in G) that transforms between the two corresponding permutations π_1, π_2 . Minimizing the number of reconfiguration steps between two configurations of the tokens (sequential/parallel token swapping) is equivalent to finding the shortest path in the Cayley graph between two given nodes corresponding to two given permutations in S_n . In fact, sequential token swapping was first studied by Cayley in 1849 [8] who (before inventing the Cayley graph) solved the problem on a clique, i.e., without any constraint on which tokens can be swapped.

Since its introduction, token swapping has been studied by many researchers in many disparate fields, from discrete mathematics [8, 25, 27, 29, 24, 28, 21] and theoretical computer science [17, 20, 13, 30, 32, 4, 23, 7, 31, 11, 18, 10, 6] to more applied fields including network engineering as mentioned earlier [2], robot motion planning [12, 26], and game theory [16].

What is the complexity of token swapping? In general, it is PSPACE-complete to find a shortest path between two given nodes in a Cayley graph defined by given generators [17]. But when the generators include transpositions (single-swap permutations) as in both sequential and parallel token swapping, $O(n^2)$ swaps always suffice [30], so the token-swapping problems are in NP. Both sequential token swapping [23] and parallel token swapping [5, 18] are known to be NP-complete on a general graph.

Sequential token swapping on general graphs is also known to be APX-hard [23], and even W[1]-hard with respect to the number of swaps [7]. For the special case of graphs with constant treewidth and constant diameter, sequential token swapping is also known to be NP-hard [7]. Additionally, for the special case of trees, but for the variant where the tokens have “weights” and “colors” sequential token swapping is known to be NP-hard [6]. From the algorithms side, there is a 4-approximation for sequential token swapping in general graphs [23]. Polynomial-time exact algorithms are known for a number of special classes of graphs including cliques [8], paths [20], cycles [17], stars [25, 24], brooms [28, 18, 6], complete bipartite graphs [30], and complete split graphs [32]. The problem is also known to be fixed parameter tractable (where the parameter is the number of swaps) on nowhere dense graphs, which includes planar graphs and graphs of bounded treewidth [7]. See also the surveys by Kim [19] and Biniáz et al. [6].

In this paper, we study the special case when the underlying graph is a tree. Sequential token swapping on a tree was first studied over thirty years ago, even before the problem was studied on general graphs. Akers and Krishnamurthy [2] studied the problem in the context of interconnection networks. Specifically, they proposed connecting processors together in a network defined by a Cayley graph, in particular a Cayley graph of transpositions corresponding to edges of a tree (what they call a *transposition tree*), so the shortest-path problem naturally arises when routing network messages. They gave an algorithm for finding short (but not necessarily shortest) paths in the resulting Cayley graphs, and characterized the diameter of the Cayley graph (and thus found optimal paths *in the worst case* over possible start/destination pairs of vertices) when the tree is a star. Follow-up work along this line attains tighter upper bounds on the diameter of the Cayley graph in this situation when the graph is a tree [27, 13, 21, 11] and develops exponential algorithms to compute the exact diameter of the Cayley graph of a transposition tree [10], though the complexity of the latter problem remains open.

Sequential token swapping on a tree is the related problem of computing the shortest-path distance between two given nodes in the Cayley graph of a transposition tree. For sequential token swapping on a tree, the literature exhibits a curious phenomenon whereby there are three 2-approximation algorithms that were all developed independently and all use completely different techniques. These algorithms are by Akers and Krishnamurthy [2] in 1989, Vaughan and Portier [29] in 1995, and Yamanaka et al. [30] in 2015. No better approximation factor than 2 is known.

Parallel token swapping was also introduced in the context of network routing: in 1994, Alon, Chung, and Graham [3] called the problem “routing permutations via matchings”. They focused on worst-case bounds for a given graph (the diameter of the Cayley graph); in particular, they proved that any n -vertex tree (and thus any n -vertex connected graph) admits a solution with less than $3n$ rounds, a bound later improved to $\frac{3}{2}n + O(\log n)$ [33]. Like sequential token swapping, computing the exact diameter of the Cayley graph of a given tree remains open.

Parallel token swapping on a tree is the related problem of computing the shortest-path distance between two given nodes in such a Cayley graph. Parallel token swapping is known to be NP-complete in bipartite maximum-degree-3 graphs, NP-complete even when restricted

to just three rounds, but polynomial-time when restricted to one or two rounds, but NP-complete again for “colored” tokens restricted to two rounds [5, 18]. Two approximation results are known: an additive approximation for paths which uses only one extra round [18], and a multiplicative $O(1)$ -approximation for the $n \times n$ grid graph [12].¹ For other special graph classes, there are tighter worst-case bounds on the diameter of the Cayley graph [3, 22, 5].

1.1 Our Results

There have been many attempts to understand token swapping on a tree, but all have fallen short of determining its actual complexity. To summarize the previously stated results for sequential token swapping on a tree, there are three known 2-approximation algorithms, and no better approximation known. There are also exact algorithms for several special cases of trees, with the most general case being a broom (a path attached to a star). From the hardness side, attempts to prove that the problem is NP-complete have led to NP-completeness proofs for more general cases. In particular, token swapping on graphs of constant treewidth and diameter is NP-hard [7], and the “weighted, colored” variant of token swapping on trees is NP-hard [6]. This leads to our first main question:

Question: Is sequential token swapping on a tree NP-complete?

This question has been implicit since sequential token swapping on a tree was first studied over 30 years ago, and the question has been explicitly stated by Biniaz et al. [6] and by Bonnet et al. [7] who conjectured that the answer is yes.

We resolve this question in the affirmative by providing a proof that sequential token swapping on a tree is NP-complete.

Next, we turn to the approximability of token swapping on a tree. The fact that there were three independently discovered 2-approximation algorithms, and nothing better is known, suggests that perhaps there is some barrier at approximation factor 2. This leads to our second main question:

Question: Is there an inherent barrier to obtaining a $(2 - \varepsilon)$ -approximation for sequential token swapping on trees?

We address this question by showing that there is indeed a restriction on the *types* of algorithms that can achieve approximation factor better than 2. To motivate the class of algorithms we rule out, it helps to examine known algorithms. Specifically, it was previously known that neither Akers and Krishnamurthy’s “happy swap” algorithm [2] nor Yamanaka et al.’s cycle algorithm [30] can possibly achieve an approximation ratio better than 2 [6]. These two algorithms share a natural property: every token t always remains within distance 1 of the shortest path from t ’s start vertex to t ’s destination vertex. A natural question is, can a better-than-2 approximation be achieved if one allows tokens to deviate from their shortest paths more, say to distance 10 or 100?

Motivated by this question, we define an *ℓ -straying* algorithm as an algorithm that never moves a token a distance more than ℓ from its shortest path. We prove a surprisingly strong limitation on ℓ -straying algorithms: any less-than-2-approximation algorithm for sequential

¹ The results of [12] are phrased in terms of motion planning for robots, and in terms of a model where an arbitrary disjoint collection of cycles can rotate one step in a round. However, the techniques quickly reduce to the model of swapping disjoint pairs of robots, so they apply to parallel motion planning as well. They show that there is always a solution within a constant factor of the obvious lower bound on the number of rounds: the maximum distance between any token’s start and destination.

token swapping on trees must in general bring a token *arbitrarily far* – an $\Omega(n^{1-\varepsilon})$ distance away – from its shortest path. That is, no ℓ -straying algorithm for $\ell = o(n^{1-\varepsilon})$ can achieve better than a 2-approximation.

The other known 2-approximation algorithm (besides [2] and [30]), is the Vaughan-Portier algorithm [29], which in fact *does* move tokens arbitrarily far from their shortest paths. That is, our result on ℓ -straying algorithms does not imply a limitation on the Vaughan-Portier algorithm. To address this, we also obtain the first proof that the Vaughan-Portier algorithm [29] is no better than a 2-approximation; the best previous lower bound for its approximation factor was $\frac{4}{3}$ [6]. Thus, none of the known algorithms or even their generalizations can improve upon the approximation factor of 2.

For parallel token swapping on a tree, less is known than for the sequential version. In particular, there is no known approximation algorithm nor is there any known hardness for tree-like graphs. Thus, the complexity of this problem is completely unclear. This leads to our third main question:

Question: What is the complexity of parallel token swapping on a tree?

We address this question by showing that parallel token swapping on a tree is NP-hard.

In summary, our results are as follows:

1. Sequential token swapping is NP-complete on trees.
2. Parallel token swapping is NP-complete on trees, even on subdivided stars.
3. Limitations on known techniques for approximating sequential token swapping on trees:
 - a) No ℓ -straying algorithm for any $\ell = O(n^{1-\varepsilon})$ can achieve better than a 2-approximation.
 - b) The Vaughan-Portier algorithm does not achieve better than a 2-approximation.

1.2 Our Techniques

NP-hardness of sequential token swapping on trees

Our NP-hardness proof for sequential token swapping on trees is our most technical and conceptually difficult result. Prior work has built towards this result by providing NP-hardness for generalizations of the problem, but there appear to be barriers against extending these techniques. In the following, we briefly review this prior work and compare it to our own.

Token swapping on trees is known to be NP-hard for the variant where tokens have weights as well as “colors” [6]. However, the use of weights and colors appears to be crucial to the reduction. Token swapping is also known to be NP-hard on graphs with treewidth 2 and diameter 6 [7]. In particular, the graph in this construction is almost a tree in the sense that if you remove a single vertex the remaining graph is a forest. However, this single vertex has very high degree and is crucial to the construction. Given the apparent barriers against extending these known approaches to token swapping on trees, we take a completely different approach.

We reduce from the *permutation generation* problem in Garey and Johnson [14, MS6] (also called the “word problem for products of symmetric groups” (WPPSG) in [15]). In comparison, the above prior work [6, 7] reduces from the vertex cover problem, and the 3-dimensional matching problem, respectively. We observe that the permutation generation problem has a similar feel to token swapping, as it can be recast in terms of a token-swapping reachability problem as follows:

3:6 Hardness of Token Swapping on Trees

Star Subsequence Token-Swapping Reachability (Star STS): Given a star graph with center vertex 0 and leaves $1, 2, \dots, m$, where vertex i initially has a token i ; given a target permutation π of the tokens; and given a sequence of swaps s_1, s_2, \dots, s_n , where $s_j \in \{1, \dots, m\}$ indicates a swap on edge $(0, s_j)$, is there a subsequence of the given swaps that realizes π ?

As a first step towards reducing from Star STS to token swapping on trees, we reduce to *weighted* token swapping on trees, where each token has a non-negative integer *weight*, and the cost of a swap is the sum of the weights of the two tokens being swapped. Our reduction contains only tokens of weight 0 or 1. That is, the tokens of weight 0 are free to move, while the tokens of weight 1 cost to move. Our reduction from Star STS to 0/1-weighted token swapping on trees is quite simple. It is presented in Section 2.

The situation becomes much more complicated when we extend this result from the 0/1-weighted setting to the unweighted setting. Now, we need to *simulate* the weight-0 tokens using unweighted tokens. This introduces several complications.

First, we will describe why weight-0 tokens are integral to our reduction from Star STS to 0/1-weighted token swapping on trees. The Star STS problem asks whether there *exists* a subsequence of swaps that realizes the target permutation π . This subsequence could contain any number of swaps. In our reduction to 0/1-weighted token swapping, the swaps from this subsequence are represented using tokens of weight 0. This way, if there is a solution to the Star STS instance, then the cost of the 0/1-weighted token swapping is the same *regardless* of how many swaps occurred in the solution to the Star STS instance. This introduces a challenge for unweighted token swapping for the following reason. For 0/1 weighted token swapping, we prove a statement of the form “if the token swapping cost is exactly K then there is a solution to the Star STS instance”, while for unweighted token swapping, we prove a statement of the form “if the token swapping cost is within a particular *range* then there is a solution to the Star STS instance”. The second statement is much more difficult to prove because we need to argue that the additional swaps in this range do not allow the tokens to move around in a clever way to admit a solution even when there is no Star STS solution. In fact, as we discuss next, natural modifications of the weighted construction *do* admit such clever ways to create counterexamples.

The most basic first attempt to remove the weights from the weighted construction is simply to replace all weight-0 tokens with unweighted tokens. This construction admits a straightforward counterexample due to the increased cost of swapping these formerly weight-0 tokens. Thus, we would like to make the contribution of the formerly weight-0 tokens negligible in comparison to the weight-1 tokens. A natural attempt is to replace each weight-1 token with a *long path* of tokens. However, as it turns out, there is a surprising and subtle counterexample to this strategy. To overcome this counterexample, we introduce a set of “padding tokens” throughout the graph whose role is to block any deviant movement of the original tokens.

The resulting proof is very involved. To give a sense of the complexity, our 0/1-weighted hardness proof fits in just a couple of pages, while our unweighted proof spans around thirty pages. Our unweighted proof is presented in Section 3.

NP-hardness of parallel token swapping on trees

We prove that parallel token swapping on trees is NP-hard, even when restricted to *subdivided stars*. This result is presented in the extended version of this paper [1]. As for sequential token swapping, we reduce from the Star STS problem.

Our construction is reminiscent of our construction for sequential token swapping, although the details differ significantly. In particular, we use the single high-degree vertex in the subdivided star as a bottleneck to limit the available parallelism. We develop “enforcement” tokens that need to swap through the high-degree vertex to force congestion at specific times. This proof’s complexity is between the weighted and unweighted sequential hardness proofs.

Limitations on known techniques for approximation algorithms

To prove that neither an ℓ -straying algorithm nor the Vaughan-Portier algorithm can achieve better than a 2-approximation, we use a problem instance that has been previously used to prove that Akers and Krishnamurthy’s and Yamanaka et al.’s algorithms cannot achieve an approximation ratio better than 2 [6]. To prove our results, we show that while there exists a solution to the instance with K swaps (for some K), (1) every ℓ -straying algorithm performs $2K$ swaps, and (2) the Vaughan-Portier algorithm performs $2K$ swaps. The existence of a solution with K swaps was already shown by [6], so it remains to show that the above algorithms require $2K$ swaps.

We emphasize that the proofs in [6] for Akers and Krishnamurthy’s and Yamanaka et al.’s algorithms, as well as our proof for the Vaughan-Portier algorithm, are for *specific* algorithms, while our proof for ℓ -straying algorithms shows limitations against a very wide class of possible algorithms. Thus, our proof for ℓ -straying algorithms requires much more general reasoning about how tokens can possibly move around the graph.

2 Weighted sequential token swapping on trees is NP-hard

In this section, we prove that weighted token swapping on a tree is NP-hard, even when the token weights are in $\{0, 1\}$. Our purpose is to introduce the general idea that is used in our main NP-completeness proof for the unweighted case given in Section 3.

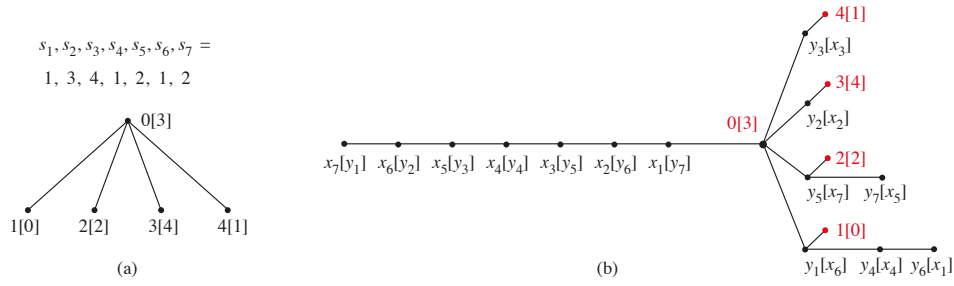
We first precisely define the decision problem *weighted sequential token swapping on trees* [6], abbreviated in this section to *weighted token swapping*. The input consists of: a tree on n vertices with distinct initial positions (vertices) and distinct target positions for the n tokens; non-negative integer weights on the tokens; and a maximum cost K . The cost of a swap is the sum of the weights of the two tokens involved. The decision problem asks whether there is a sequence of swaps that moves all the tokens to their target positions and such that the sum of the costs of the swaps is at most K .

It is not clear whether the weighted token swapping problem lies in NP; however, it is in NP if K is given in unary.²

We prove that weighted token swapping is NP-hard when K is given in unary. The reduction uses Star STS. In order to distinguish tokens and vertices in the original star from those in the tree that we construct, we will call the tokens of the Star STS instance *items* and we will call the leaves *slots*. Then the instance of Star STS consists of: a star with center 0 and slots $1, \dots, m$, each of which initially has an item of its same label; a permutation π of the items; and a sequence s_1, \dots, s_n of slots that specify the allowed swaps. Figure 1(a) shows an example input for $m = 4$ slots, 5 items and a sequence of length $n = 7$.

► **Theorem 1.** *Weighted token swapping on trees is NP-hard.*

² Consider a minimum length swap sequence of weight at most K . The number of swaps involving a nonzero weight token is at most K . Because the sequence has minimum length, it can be shown that no two zero-weight tokens swap more than once. Thus the sequence has length at most $K + n^2$ and provides a polynomial-size certificate, showing that the problem is in NP.



■ **Figure 1** (a) An instance of Star STS with $m = 4$ slots and a sequence of length $n = 7$; notation $a[b]$ indicates that token a is initially at this vertex and token b should move to this vertex. This instance has no solution because item 4 should move to slot 3, which is possible only if slot 3 appears after slot 4 in the sequence. (b) The corresponding instance of weighted token swapping with the ordering gadget on the left and $m = 4$ slot gadgets attached to the root, each with a nook vertex shown in red. After swapping item token 0 at the root with token y_1 in the first (bottom) slot gadget there is an opportunity to swap tokens 0 and 1 for free along the nook edge of the first slot gadget, before moving y_1 to its target position at the end of the ordering gadget and moving x_1 to its target position at the end of the first slot gadget.

Proof. Suppose we are given an instance of Star STS as described above. We may assume without loss of generality that every slot appears in the sequence (otherwise remove that slot from the problem) and that no slot appears twice in a row in the sequence.

Construct a tree with a root, an *ordering gadget* which is a path of length n attached to the root, and m *slot gadgets* attached to the root. Slot gadgets are defined below. See Figure 1(b) where the ordering gadget of length $n = 7$ appears on the left and there are $m = 4$ slot gadgets attached to the root. We picture the tree with the root in the middle, and use directions left/right as in the figure.

Let n_i be the number of occurrences of slot i in the input sequence. Observe that $\sum_{i=1}^m n_i = n$. *Slot gadget* i consists of a path of n_i vertices, plus an extra leaf attached to the leftmost vertex of the path. This extra leaf is called the *nook* of the slot gadget. The m nooks and the root are in one-to-one correspondence with the slots and the center of the original star (respectively), and we place *item tokens* at these vertices whose names, initial positions, and final positions correspond exactly to those of the items of the input star. These item tokens are given a weight of 0.

There are $2n$ additional *non-item tokens* x_1, \dots, x_n and y_1, \dots, y_n . These all have weight 1. The x_j 's are initially placed along the ordering gadget path, in order, with x_1 at the right and x_n at the left. The ordering path is also the target position of the y_j 's in reverse order with y_1 at the left and y_n at the right.

Suppose slot i appears in the sequence as $s_{j_1}, s_{j_2}, \dots, s_{j_{n_i}}$ with indices in order $j_1 < j_2 < \dots < j_{n_i}$. Then tokens $y_{j_1}, y_{j_2}, \dots, y_{j_{n_i}}$ are initially placed along the path of slot gadget i , in order with smallest index at the left. The path of slot gadget i is also the target position of the tokens $x_{j_1}, x_{j_2}, \dots, x_{j_{n_i}}$ in reverse order with smallest index at the right.

Consider, for each non-item token x_j or y_j , the distance from its initial location to its target location. This is a lower bound on the cost of moving that token. We set the max cost K to the sum of these lower bounds. This guarantees that every x_j or y_j only travels along its shortest path. Observe that this reduction takes polynomial time. We now prove that a YES instance of Star STS maps to a YES instance of token swapping and vice versa.

YES instance of Star STS. Suppose the Star STS instance has a solution. The “intended” solution to the token swapping instance implements each s_j for $j = 1, \dots, n$ as follows. Suppose $s_j = i$. By induction on j , we claim that token y_j will be in the leftmost vertex of slot gadget i when it is time to implement s_j . Swap token y_j with the item token t currently at the root. Then item token t has the opportunity to swap for free with the item token in nook i . We perform this free swap if and only if swap s_j was performed in the solution to Star STS. Next, swap tokens y_j and x_j – we claim by induction that x_j will be in the rightmost vertex of the ordering gadget. Finally, move token y_j to its target position in the ordering gadget, and move x_j to its target position in slot gadget i . It is straightforward to verify the induction assumptions. Every x_j and y_j moves along its shortest path so the cost of the solution is equal to the specified bound K .

YES instance of weighted token swapping. Suppose the weighted token swapping instance has a solution with at most K swaps. Because K is the sum of the distances of the non-item tokens from their target positions, each x_j and y_j can only move along the shortest path to its target position. Token x_j must move into the root before x_{j+1} , otherwise they would need to swap before that, which means moving x_j the wrong way along the ordering gadget path. Similarly, y_j must move into the root before y_{j+1} , otherwise they need to swap after that, which means moving y_{j+1} the wrong way.

Furthermore, x_j must move into the root before y_{j+1} otherwise the ordering gadget would contain x_j, \dots, x_n , and y_1, \dots, y_j , a total of $n + 1$ tokens, which is more tokens than there are vertices in the ordering gadget.

We also claim y_j must move into the root before x_{j+1} . The nooks can only contain item tokens, since no x_j or y_j can move into a nook. This accounts for every item token except for one “free” item token. Now suppose x_{j+1} moves into the root before y_j . Then the ordering gadget contains x_{j+2}, \dots, x_n , and y_1, \dots, y_{j-1} , a total of $n - 2$ tokens. Even if the free item token is in the ordering gadget, there are not enough tokens to fill the ordering gadget.

Thus the x_j 's and y_j 's must use the root in order, first x_1 and y_1 in some order, then x_2 and y_2 in some order, etc. Finally, we examine the swaps of item tokens. A swap between two item tokens can only occur when the free item token is at the parent of a nook. Suppose this happens in slot gadget i . Then some token y_j must have left the slot gadget, and the corresponding token x_j has not yet entered the slot gadget, which means that neither of the tokens y_{j+1} or x_{j+1} has moved into the root. This implies that any swap of item tokens is associated with a unique $s_j = i$, and such swaps must occur in order of j , $1 \leq j \leq n$. Thus the swaps of item tokens can be mimicked by swaps in the original Star STS sequence, and the Star STS instance has a solution. ◀

3 Sequential token swapping on trees is NP-complete

Recall that the token swapping problem is a decision problem: given a tree with initial and target positions of the tokens, and given a non-negative integer K , can the tokens be moved from their initial to their target positions with at most K swaps. Membership in NP is easy to show: any problem instance can always be solved with a quadratic number of swaps by repeatedly choosing a leaf and swapping its target token to it. Thus, a certificate can simply be the list of swaps to execute. Here we give an overview of the construction for the unweighted case; however, most of the proof is left to the full version [1].

3.1 Overview

We reduce from Star STS and follow the same reduction plan as for the weighted case, building a tree with an ordering gadget and m slot gadgets, each with a nook for an item token. However, there are several challenges. First of all, the swaps with item tokens are no longer free, so we need to take their number into account. Secondly, we cannot know the number of item token swaps precisely since it will depend on the number of swaps (between 0 and n) required by the original Star STS instance. Our plan is to make this “slack” n very small compared to the total number of swaps needed for the constructed token swapping instance. To do this, we will make the total number of swaps very big by replacing each of the non-item tokens x_j and y_j by a long path of non-item tokens called a “segment.” This raises further difficulties, because nothing forces the tokens in one segment to stay together, which means that they can sneak around and occupy nooks, freeing item tokens to swap amongst themselves in unanticipated ways. To remedy this we add further “padding segments” to the construction. The construction details are given in Section 3.2.

Proving that our unweighted construction is correct is much more involved than in the weighted case. There are two parts to the proof.

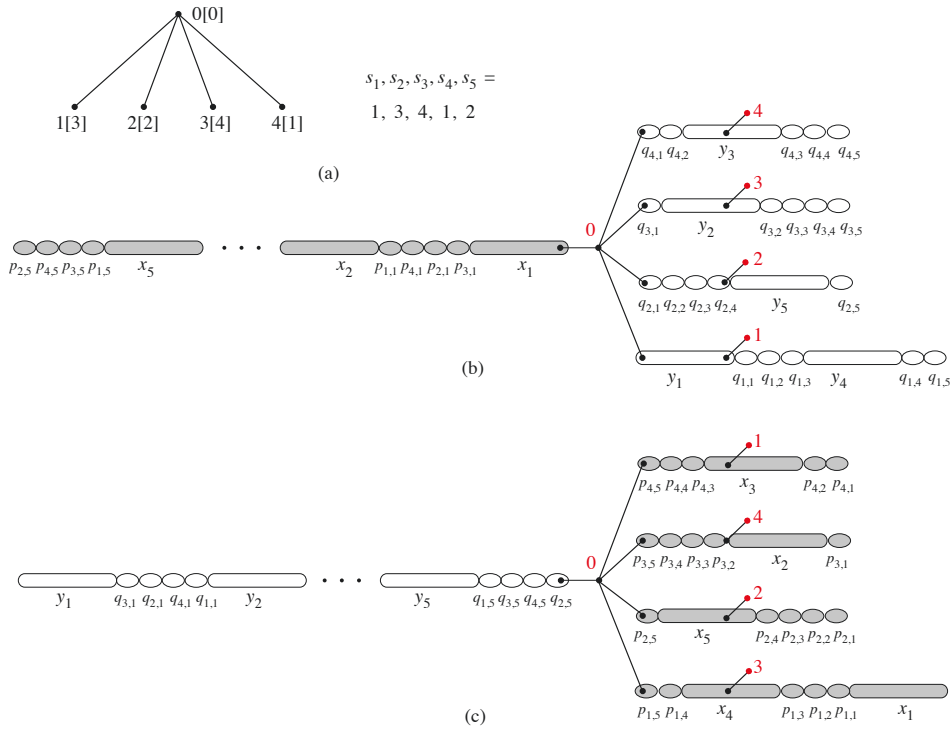
Part 1: YES instance of Star STS \rightarrow YES instance of token swapping. In the full version [1] we show that if there is a solution to an instance of Star STS then the constructed token swapping instance can be solved with at most K swaps (where K will be specified in the construction). This solution uses exactly $H = K - n$ swaps to get the non-item tokens to their target positions, and uses between 0 and n additional swaps to get the item tokens to their target positions. Note that H counts swaps of both item and non-item tokens and is more than just the cost of moving each non-item token along its shortest path.

Part 2: YES instance of token swapping \rightarrow YES instance of Star STS. In the full version [1] we show that if the constructed token swapping instance can be solved with at most K swaps then the original Star STS instance has a solution. We show that H swaps are needed to get the non-item tokens to their destinations. We then show that with only n remaining swaps, the motion of item tokens is so constrained that they must behave “as intended” and therefore correspond to swaps in the original Star STS instance.

3.2 Construction of token swapping instance

Suppose we have an instance of Star STS where the star has center 0 and leaves $1, \dots, m$ and each vertex has a token of its same label. We have a permutation π of the tokens with $\pi(0) = 0$ and a sequence s_1, \dots, s_n with $s_j \in \{1, \dots, m\}$ that specifies the allowed swaps. As in the weighted case in Section 2, we will refer to the tokens of the Star STS instance as *items* and the leaves as *slots*. We assume without loss of generality that every slot appears in the sequence (otherwise remove that slot from the problem) and that no slot appears twice in a row in the sequence.

Construct a tree as in the weighted case except that each individual x_j and y_j is replaced by a sequence of k vertices (and tokens) with $k = (mn)^c$ for a large constant c , to be set later. Each such sequence of length k is called a *big segment*. Refer to Figure 2 where the tree is drawn with the root in the middle, the ordering gadget to the left, and the slot gadgets to the right. We will refer to left and right as in the figure. The target ordering of tokens within a big segment behaves as though the segment just slides along the shortest path to its target, i.e., the left to right order of tokens in a segment is the same in the initial and the final configurations.



■ **Figure 2** (a) An instance of Star STS with $m = 4$ slots and a sequence of length $n = 5$. (b) The corresponding instance of token swapping with the initial token positions. The root has item token 0 (coloured red). The ordering gadget lies to the left of the root. There are 4 slot gadgets to the right of the root. A long oval indicates a big segment of length k , and a short oval indicates a padding segment of length $k' = k/n^8$. Each nook vertex (coloured red) is attached to the k^{th} vertex from the root along the slot gadget path. Item tokens are coloured red; non-slot tokens are in the segment ovals coloured gray; and slot tokens are in the segment ovals coloured white. (c) The target token positions. In the first round of the “intended” solution, big segments y_1 and x_1 first change places. As y_1 moves left, item token 0 moves to nook parent 1 where it may swap with item token 1. As x_1 moves right, the item token moves back to the root. Then segment y_1 moves to the far left of the ordering gadget and x_1 moves to the far right of the first slot gadget. Next, padding segments $p_{3,1}$ and $q_{3,1}$ change places across the root and move to their target locations; then $p_{2,1}$ and $q_{2,1}$; then $p_{4,1}$ and $q_{4,1}$; and finally $p_{1,1}$ and $q_{1,1}$. Note the ordering $q_{3,1}, q_{2,1}, q_{4,1}, q_{1,1}$ of padding segments that lie to the right of y_1 in the final configuration – $q_{1,1}$ is last because $s_1 = 1$ and $q_{3,1}$ is first because $s_2 = 3$.

The nook vertex in each slot gadget is attached to the vertex at distance k from the root in the slot gadget, and this vertex is called the *nook parent*. The edge between the nook vertex and the nook parent is called the *nook edge*, see Figure 2. In the “intended” solution, the big segments leave the slot gadgets and enter the ordering gadget in the order y_1, \dots, y_n .

Although we will not give details, the construction so far allows “cheating” via interference between slot gadgets. To prevent this, we add a total of $2nm$ padding segments each of length $k' = k/n^8$. The intuition is in the “intended” solution, after y_j enters the ordering gadget, one padding segment from each slot gadget will enter the ordering gadget, and then y_{j+1} will do so. We now give the details of the padding segments in the initial/final configurations of the slot/ordering gadgets. In the initial configuration there are nm padding segments $q_{i,j}, i = 1, \dots, m, j = 1, \dots, n$ in the slot gadgets, and nm padding segments $p_{i,j}, i = 1, \dots, m, j = 1, \dots, n$ in the ordering gadget. In the final configuration, the $q_{i,j}$'s

are in the ordering gadget and the $p_{i,j}$'s are in the slot gadgets. In the initial configuration, slot gadget i , $i = 1, \dots, m$, contains n padding segments $q_{i,j}$, $j = 1, \dots, n$. They appear in order from left to right with big segments mixed among them. Specifically, if $y_{j'}$ is a big segment in slot gadget i then $y_{j'}$ appears just before $q_{i,j'}$. In the final configuration of the ordering gadget there are m padding segments after each of the n big segments. The padding segments after y_j are $q_{i,j}$, $i = 1, \dots, m$. They appear in a particular left-to-right order: $q_{s_j,j}$ is last, $q_{s_{j+1},j}$ is first, and the others appear in order of index i . Within one padding segment, the left-to-right ordering of tokens is the same in the initial and final configurations.

The initial configuration of the ordering gadget is obtained from the final configuration by: reversing (from left to right) the pattern of big segments and padding segments; changing y 's to x 's, and changing q 's to p 's. Similarly, the final configuration of slot gadget i is obtained from the initial configuration of slot gadget i in the same way.

Let A be the set of non-item tokens, and for any token t , let d_t be the distance between t 's initial and target positions. To complete the reduction, we will set H to be $\frac{1}{2} \sum_{t \in A} (d_t + 1)$ and set the bound K to be $H + n$. The decision question is whether this token swapping instance can be solved with at most K swaps. The reduction takes polynomial time.

The remainder of the proof can be found in the full version [1].

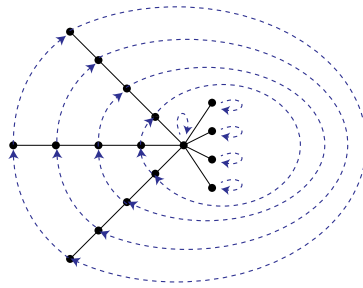
4 Known techniques preclude approximation factors less than 2

Previous results about sequential token swapping on trees include three different polynomial time 2-approximation algorithms and some lower bounds on approximation factors. The three algorithms all have the property that if a token at a leaf is at its destination (i.e., it is a “happy leaf token”) then the algorithm will not move it. Biniáz et al. [6] proved that any algorithm with this property has a (worst case) approximation factor at least $\frac{4}{3}$. They also proved via ad-hoc arguments that the approximation factor is exactly 2 for two of the known 2-approximation algorithms (the “Happy Swap Algorithm” and the “Cycle Algorithm”). For the third 2-approximation algorithm, the Vaughan-Portier algorithm, they could not prove a lower bound better than $\frac{4}{3}$.

We prove that the Vaughan-Portier algorithm has approximation factor exactly 2. We also extend the approximation lower bounds of Biniáz et al. by proving that the approximation factor is at least 2 for a larger family of algorithms. We formalize this family as follows. For token t let P_t be the path from t 's initial position to its final position. A sequence of token swaps is **ℓ -straying** if at all intermediate points along the sequence, every token t is within distance ℓ of the last vertex of P_t that it has reached up to this point. A token swapping algorithm is **ℓ -straying** if it produces ℓ -straying sequences.

Both approximation lower bounds will be proved for the same family of trees that was used by Biniáz et al. [6]. For any k and any odd b we define a tree $T_{k,b}$ together with initial and final positions of tokens. The tree $T_{k,b}$ has b paths of length k attached to a central vertex c , and a set L of k leaves also attached to c . See Figure 3. The tokens at c and L are **happy** – they are at their final positions. The tokens in branch i , $0 \leq i \leq b - 1$, have their final positions in branch $i + 1$, addition modulo b , with the initial and final positions equally far from the center c .

Biniáz et al. [6] proved that the optimum number of swaps for $T_{k,b}$ is at most $(b + 1) \binom{k+1}{2} + 2k$. The solution repeatedly exchanges the tokens in branch i , $0 \leq i \leq b - 1$, modulo b , with the tokens at L . The first exchange moves the tokens initially at L into branch 0, and the $(b + 1)^{\text{st}}$ exchange moves those tokens back to L . In the full version [1] we prove that for $T_{k,b}$ the approximation factor is not better than 2 for ℓ -straying algorithms and for the Vaughan-Portier algorithm.



■ **Figure 3** Tree $T_{k,b}$ with $b = 3$ branches each of length $k = 4$, and with $k = 4$ leaves attached to the center node. The dashed arrows go from a token's initial to final position. The figure is from [6].

5 Open Problems

Many interesting related problems remain open. For sequential token swapping on trees, where is the divide between NP-complete and polynomial-time? Our reduction's tree is a subdivided star (as for parallel token swapping) with the addition of one extra leaf (the nook) per path. By contrast, there is a polynomial-time algorithm for the case of a broom (a star with only one edge subdivided) [28, 18, 6]. What about a star with two subdivided edges?

For parallel token swapping, even the case of a single long path is open. Kawahara et al. [18] gave an additive approximation algorithm that uses at most one extra round. Is there an optimal algorithm, or is parallel token swapping NP-hard on paths?

There are also open problems in approximation algorithms. In parallel token swapping, we know that there is no PTAS [18]. Is there an $O(1)$ -approximation for trees or general graphs? For sequential token swapping, there is a 4-approximation algorithm [23]. Is 4 a lower bound on the approximation factor of this algorithm? Is 4-approximation the best possible for general graphs?

Although this paper focused on the best reconfiguration sequence, much research is devoted to understanding the worst-case behavior for a given graph; is it NP-hard to determine the diameter of the Cayley graph, i.e., the maximum number of reconfiguration steps that can be required for any pair of token configurations? This problem is open for both sequential token swapping (implicit in [10]) and parallel token swapping [3].

References

- 1 Oswin Aichholzer, Erik D. Demaine, Matias Korman, Jayson Lynch, Anna Lubiw, Zuzana Masárová, Mikhail Rudoy, Virginia Vassilevska Williams, and Nicole Wein. Hardness of token swapping on trees. *CoRR*, abs/2103.06707, 2021. [arXiv:2103.06707](https://arxiv.org/abs/2103.06707).
- 2 Sheldon B. Akers and Balakrishnan Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, 1989.
- 3 Noga Alon, F. R. K. Chung, and R. L. Graham. Routing permutations on graphs via matchings. *SIAM Journal on Discrete Mathematics*, 7(3):513–530, 1994. [doi:10.1137/S0895480192236628](https://doi.org/10.1137/S0895480192236628).
- 4 Amihod Amir and Benny Porat. On the hardness of optimal vertex relabeling and restricted vertex relabeling. In *Symposium on Combinatorial Pattern Matching (CPM)*, volume 9133 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2015.
- 5 Indranil Banerjee and Dana Richards. New results on routing via matchings on graphs. In R. Klasing and M. Zeitoun, editors, *Proceedings of the 21st International Symposium on Fundamentals of Computation Theory*, volume 10472 of *Lecture Notes in Computer Science*, 2017.

- 6 Ahmad Biniiaz, Kshitij Jain, Anna Lubiw, Zuzana Masárová, Tillmann Miltzow, Debajyoti Mondal, Anurag Murty Naredla, Josef Tkadlec, and Alexi Turcotte. Token swapping on trees. arXiv preprint, 2019. [arXiv:1903.06981](https://arxiv.org/abs/1903.06981).
- 7 Édouard Bonnet, Tillmann Miltzow, and Paweł Rzażewski. Complexity of token swapping and its variants. *Algorithmica*, 80(9):2656–2682, 2018.
- 8 Arthur Cayley. LXXVII. Note on the theory of permutations. *Philosophical Magazine Series 3*, 34(232):527–529, 1849.
- 9 Arthur Cayley. Desiderata and suggestions: No. 2. the theory of groups: graphical representation. *American Journal of Mathematics*, 1(2):174–176, 1878.
- 10 Bhadrachalam Chitturi and Priyanshu Das. Sorting permutations with transpositions in $O(n^3)$ amortized time. *Theoretical Computer Science*, 766:30–37, 2019. doi:10.1016/j.tcs.2018.09.015.
- 11 Bhadrachalam Chitturi and Indulekha T S. Sorting permutations with a transposition tree. arXiv preprint, 2018. [arXiv:1811.07443](https://arxiv.org/abs/1811.07443).
- 12 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Christian Scheffer, and Henk Meijer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. In *Proceedings of the 34th International Symposium on Computational Geometry*, pages 29:1–29:15, June 2018.
- 13 Ashwin Ganesan. An efficient algorithm for the diameter of Cayley graphs generated by transposition trees. *International Journal of Applied Mathematics*, 42(4), 2012. [arXiv:1202.5888](https://arxiv.org/abs/1202.5888).
- 14 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, 1979.
- 15 Michael R. Garey, David S. Johnson, Gary L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic Discrete Methods*, 1(2):216–227, 1980.
- 16 Laurent Gourvès, Julien Lesca, and Anaëlle Wilczynski. Object allocation via swaps along a social network. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 213–219, 2017. URL: <https://www.ijcai.org/Proceedings/2017/0031.pdf>.
- 17 Mark R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985.
- 18 Jun Kawahara, Toshiki Saitoh, and Ryo Yoshinaka. The time complexity of permutation routing via matching, token swapping and a variant. *Journal of Graph Algorithms and Applications*, 23(1):29–70, 2019.
- 19 Dohan Kim. Sorting on graphs by adjacent swaps using permutation groups. *Computer Science Review*, 22:89–105, 2016.
- 20 Donald Ervin Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, 2nd edition, 1998.
- 21 Benjamin Kraft. Diameters of Cayley graphs generated by transposition trees. *Discrete Applied Mathematics*, 184:178–188, 2015.
- 22 Wei-Tian Li, Linyuan Lu, and Yiting Yang. Routing numbers of cycles, complete bipartite graphs, and hypercubes. *SIAM Journal on Discrete Mathematics*, 24(4):1482–1494, 2010. doi:10.1137/090776317.
- 23 Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and hardness of token swapping. In *Proceedings of the 24th Annual European Symposium on Algorithms*, volume 57 of *LIPICs*, 2016.
- 24 Igor Pak. Reduced decompositions of permutations in terms of star transpositions, generalized Catalan numbers and k -ary trees. *Discrete Mathematics*, 204(1-3):329–335, 1999.
- 25 Frederick J. Portier and Theresa P. Vaughan. Whitney numbers of the second kind for the star poset. *European Journal of Combinatorics*, 11(3):277–288, 1990.

- 26 Pavel Surynek. Multi-agent path finding with generalized conflicts: An experimental study. In Jaap van den Herik, Ana Paula Rocha, and Luc Steels, editors, *Revised Selected Papers from the 11th International Conference on Agents and Artificial Intelligence*, pages 118–142, February 2019.
- 27 Theresa P. Vaughan. Bounds for the rank of a permutation on a tree. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 10:65–81, 1991.
- 28 Theresa P. Vaughan. Factoring a permutation on a broom. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 30:129–148, 1999.
- 29 Theresa P. Vaughan and Frederick J. Portier. An algorithm for the factorization of permutations on a tree. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 18:11–31, 1995.
- 30 Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theoretical Computer Science*, 586:81–94, 2015.
- 31 Katsuhisa Yamanaka, Takashi Horiyama, J. Mark Keil, David Kirkpatrick, Yota Otachi, Toshiki Saitoh, Ryuhei Uehara, and Yushi Uno. Swapping colored tokens on graphs. *Theoretical Computer Science*, 729:1–10, 2018.
- 32 Gaku Yasui, Kouta Abe, Katsuhisa Yamanaka, and Takashi Hirayama. Swapping labeled tokens on complete split graphs. *Inf. Process. Soc. Japan. SIG Tech. Rep*, 2015(14):1–4, 2015.
- 33 Louxin Zhang. Optimal bounds for matching routing on trees. *SIAM Journal on Discrete Mathematics*, 12(1):64–77, 1999. doi:10.1137/S0895480197323159.

Tight Bounds for Online Matching in Bounded-Degree Graphs with Vertex Capacities

Susanne Albers 

Department of Computer Science, Technische Universität München, Germany

Sebastian Schubert¹  

Department of Computer Science, Technische Universität München, Germany

Abstract

We study the b -matching problem in bipartite graphs $G = (S, R, E)$. Each vertex $s \in S$ is a server with individual capacity b_s . The vertices $r \in R$ are requests that arrive online and must be assigned instantly to an eligible server. The goal is to maximize the size of the constructed matching. We assume that G is a (k, d) -graph [19], where k specifies a lower bound on the degree of each server and d is an upper bound on the degree of each request. This setting models matching problems in timely applications.

We present tight upper and lower bounds on the performance of deterministic online algorithms. In particular, we develop a new online algorithm via a primal-dual analysis. The optimal competitive ratio tends to 1, for arbitrary $k \geq d$, as the server capacities increase. Hence, nearly optimal solutions can be computed online. Our results also hold for the vertex-weighted problem extension, and thus for AdWords and auction problems in which each bidder issues individual, equally valued bids.

Our bounds improve the previous best competitive ratios. The asymptotic competitiveness of 1 is a significant improvement over the previous factor of $1 - 1/e^{k/d}$, for the interesting range where $k/d \geq 1$ is small. Recall that $1 - 1/e \approx 0.63$. Matching problems that admit a competitive ratio arbitrarily close to 1 are rare. Prior results rely on randomization or probabilistic input models.

2012 ACM Subject Classification Theory of computation \rightarrow Online algorithms

Keywords and phrases online algorithms, deterministic algorithms, primal-dual analysis, b -matching, bounded-degree graph, variable vertex capacities, unweighted matching, vertex-weighted matching

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.4

Related Version *Full Version*: <https://arxiv.org/abs/2206.15336>

1 Introduction

Maximum matching is a fundamental problem in computer science. In a seminal paper Karp, Vazirani and Vazirani [15] introduced online matching in bipartite graphs $G = (S \cup R, E)$. The vertices of S are known in advance, while the vertices of R (requests) arrive one by one and must be matched immediately to an eligible partner. The b -matching problem is a generalization where the vertices of S (servers) have capacities and may be matched multiple times, see e.g. [14]. Online bipartite matching and capacitated extensions have received tremendous research interest over the past 30 years. In this paper we study the b -matching problem in bounded-degree graphs, defined in [19]. We assume that there is a lower bound on the degree of each server $s \in S$, meaning that there is a certain demand for each server. Furthermore we assume that there is an upper bound on the degree of each $r \in R$, i.e. each request can only be assigned to a subset of the servers. This setting models matching problems in many timely applications, as we will describe below.

¹ Corresponding author



4:2 Tight Bounds for Online b -Matching in Bounded-Degree Graphs

More formally, we investigate the following problem. Again, let $G = (S \cup R, E)$ be a bipartite graph, where the vertices of S are servers and the vertices of R are requests. The set S is known in advance. Each server $s \in S$ has an individual capacity $b_s \in \mathbb{N}$, indicating that the server can be matched with up to b_s requests. The vertices of R arrive online, one by one. Whenever a new request $r \in R$ arrives, its incident edges are revealed. The request has to be matched immediately and irrevocably to an eligible server, provided that there is one. The goal is to maximize the number of matching edges. We will also examine the *vertex-weighted* problem extension, where additionally each server $s \in S$ has a weight w_s and the value of every matching edge incident to s is multiplied by w_s . Now the goal is to maximize the total weight of the constructed matching.

We assume that G is a (k, d) -graph, defined by Naor and Wajc [19], where k and d are positive integers. Each server $s \in S$ has a degree $d(s) \geq k \cdot b_s$. Each request $r \in R$ has a degree $d(r) \leq d$. Naor and Wajc [19] defined these graphs for the general AdWords problem. Note that the inequality $d(s) \geq k \cdot b_s$ expresses a degree bound in terms of the server capacity. This is essential. As we shall see, the performance of algorithms depends on the degrees $d(s)$ as a function of b_s . A degree bound independent of b_s is vacuous for larger b_s . Also, a company operating a high-capacity server expects the server to be attractive and a potential host for a large number of requests. Otherwise it might be beneficial to reduce the server capacity.

The best results will be obtained if $k \geq d$. In this case the average demand for each server slot is high, compared to the number of servers a request can be assigned to. This setting is also very relevant in applications. We will assume that $d \geq 2$. If $d = 1$, any GREEDY algorithm constructs an optimal matching. We remark that (k, d) -graphs are loosely related to d -regular graphs in which each vertex has a degree of exactly d and a capacity of 1. This graph class has been studied extensively in computer science and discrete mathematics, see e.g. [5, 6, 7, 10, 20].

The b -matching problem in (k, d) -graphs models many problems in modern applications, cf. [4, 12, 19]. The following description also addresses the degree constraints.

Video content delivery, web hosting, remote data storage: Consider a collection of servers in a video content delivery network, a web hosting provider, or a remote data storage service. A sequence of clients arrives, each with a request that videos be streamed, web pages be hosted, or data be stored. Based on the servers' geographic distribution, average performance, technology used or pricing policies, each request can only be hosted at a small subset of the servers or server locations. Each server has a large capacity and is well suited to service a huge number of requests in the arriving client sequence.

Job scheduling: Consider a collection of compute servers, each with certain capabilities, located for instance in a data center. Over a time horizon jobs arrive, requesting service. Based on computing demands, expected response time, hardware and software requirements, each job can only be executed on a subset of the servers. During the given time horizon, each server can process a large number of jobs and is a suitable platform to execute very many of the incoming jobs.

AdWords and ad auctions: Consider a search engine company or digital advertising platform. There is a set of advertisers, each with a daily budget, who wish to link their ads to users of the search engine/digital platform and issue respective bids. The users arrive online and must be allocated instantly to the advertisers. Based on his search keywords, browsing history and possible profile, each user is interesting to a small set of advertisers. Each advertiser has a decent budget and targets a large population of the users. Obviously, in this application the advertisers correspond to the servers and the users are the incoming

requests. The b -matching problem models the basic setting where the bids of all advertisers are either 0 or 1. The vertex-weighted extension captures the scenario where all the bids of an advertiser $s \in S$ have a value of 0 or w_s . These base cases are also studied in recent work by Vazirani [21].

We analyze the performance of online algorithms using competitive analysis. Given an input graph G , let $\text{ALG}(G)$ denote the size (or weight) of the matching constructed by an online algorithm ALG . Let $\text{OPT}(G)$ be the corresponding value of an optimal offline algorithm OPT . Algorithm ALG is c -competitive if $\text{ALG}(G) \geq c \cdot \text{OPT}(G)$ holds, for all G . In our analyses we will focus on bipartite (k, d) -graphs G .

Related work. As mentioned above, Karp et al. [15] introduced online matching in bipartite graphs, in which every vertex has a capacity of 1. The best competitive ratio of deterministic online algorithms is equal to $1/2$. Karp et al. proposed a randomized RANKING algorithm that achieves an optimal competitive ratio of $1 - 1/e \approx 0.63$. Aggarwal et al. [1] defined online vertex-weighted bipartite matching and devised a $(1 - 1/e)$ -competitive algorithm.

Kalyanasundaram and Pruhs [14] investigated the b -matching problem if all servers have equal capacity, i.e. $b_s = b$ for all $s \in S$. They presented a deterministic BALANCE algorithm that matches a new request to an adjacent server whose current load is smallest. BALANCE achieves an optimal competitive ratio of $1 - 1/(1 + 1/b)^b$. As b grows, the latter expression tends from below to $1 - 1/e$. Grove et al. [12] and Chaudhuri et al. [4] studied b -matchings with a different objective. At any time an algorithm must maintain a matching between the requests that have arrived so far and the servers. The goal is to minimize the total number of switches, reassigning requests to different servers.

The AdWords problem was formally defined by Mehta et al. [18]. They presented a deterministic online algorithm that achieves a competitive ratio of $1 - 1/e$, under the *small-bids assumption* where the bids are small compared to the advertisers' budgets. No randomized algorithm can obtain a better competitive factor. Buchbinder et al. [3] examined a setting where the degree of each incoming user is upper bounded by d and gave an algorithm with a competitive ratio of nearly $1 - (1 - 1/d)^d$. Azar et al. [2] showed that this ratio is best possible, also for randomized algorithms. The expression $1 - (1 - 1/d)^d$ is always greater than $1 - 1/e$ but approaches the latter value as d increases.

The class of (k, d) -graphs was defined by Naor and Wajc [19], who studied online bipartite matching and the AdWords problem. They proposed an algorithm HIGHDEGREE that matches a new request to an available neighbor of highest current degree. Naor and Wajc proved that HIGHDEGREE and generalizations attain a competitive factor of $1 - (1 - 1/d)^k$. This ratio holds for online bipartite matching and the vertex-weighted extension, where all vertices have a capacity of 1. Furthermore, it holds for AdWords with equal bids per bidder. For AdWords with arbitrary bids, the ratio is $(1 - R_{\max})(1 - (1 - 1/d)^k)$, where R_{\max} is the maximum ratio between the bid of any bidder and its total budget. Naor and Wajc showed that no deterministic online algorithm for bipartite matching can achieve a competitive ratio greater than $1 - (1 - 1/d)^k$ if $k \geq d$. For the general AdWords problem, they proved an upper bound of $(1 - R_{\max})(1 - (1 - 1/d)^{k/R_{\max}})$ if $k \geq d$. For increasing k/d , the expression $1 - (1 - 1/d)^k$ tends to 1. For $k \approx d$ increasing, it approaches again $1 - 1/e$.

Cohen and Wajc [5] studied online bipartite matching in d -regular graphs and developed a randomized algorithm with a competitive ratio of $1 - O(\sqrt{\log d/d})$, which tends to 1 as d increases.

Online bipartite matching and the AdWords problem have also been examined in stochastic input models. A random permutation of the vertices of R may arrive. Alternatively, the vertices of R are drawn i.i.d. from a known or unknown distribution. For online bipartite

matching, the best online algorithms currently known achieve competitive ratios of 0.696 and 0.706 [13, 16]. The best possible performance ratios are upper bounded by 0.823 [17], and hence bounded away from 1. For AdWords, $(1 - \varepsilon)$ -competitive algorithms are known, based on the small-bids assumption [8, 9].

Our contributions. We present a comprehensive study of the b -matching problem in (k, d) -graphs. Specifically, we develop tight lower and upper bounds on the performance of deterministic online algorithms. The optimal competitive ratio tends to 1, for any choice of k and d with $k \geq d$, as the server capacities increase.

First, in Section 2 we investigate the setting that all servers have the same capacity, i.e. $b_s = b$ for all $s \in S$. We develop an optimal online algorithm `WEIGHTEDASSIGNMENT` via a primal-dual analysis. The resulting strategy is simple. Associated with each server of current load l and current degree δ is a value $V(l, \delta)$. An incoming request is assigned to an eligible server for which the increment $V(l, \delta + 1) - V(l, \delta)$ is maximized. The values $V(l, \delta)$ can be calculated in a preprocessing step and retrieved by table lookup when the requests of R are served. The best values $V(l, \delta)$, for variable l and δ , are determined using recurrence relations. Solving them is non-trivial because two parameters are involved.

We prove that `WEIGHTEDASSIGNMENT` achieves a competitive ratio of c^* , where

$$c^* = 1 - \frac{1}{b} \left(\sum_{i=1}^b i \binom{kb}{b-i} \frac{1}{(d-1)^{b-i}} \right) \left(1 - \frac{1}{d} \right)^{kb}.$$

This is a slightly complex expression, but it is exact in all terms. In Section 3 we prove that no deterministic online algorithm can attain a competitive ratio greater than c^* , for any choices of k and d that fulfill $k \geq d$.

In Section 4 we consider two generalizations. We assume that each server $s \in S$ has an individual capacity b_s and adapt `WEIGHTEDASSIGNMENT`. As for the competitive factor, in c^* the capacity b has to be replaced by $b_{\min} := \min_{s \in S} b_s$. The resulting competitiveness is again optimal for $k \geq d$. Furthermore, we study the vertex-weighted problem extension and again adjust our algorithm. The competitive ratios are identical to those in the unweighted setting, for uniform and variable server capacities. Our results also hold for the AdWords problem where bidders issue individual, equally valued bids.

In Section 5 we analyze the optimal competitive ratio c^* . We prove that it tends to 1, for any $k \geq d$, as b increases. Furthermore, we show that it is strictly increasing in b . The analyses are involved and make non-trivial use of Gauss hypergeometric functions.

A strength of our results is that the optimal competitiveness tends to 1, for increasing server capacities. Hence almost optimal solutions can be computed online. For the AdWords problem, high server capacities correspond to the small-bids assumption. Remarkably, in this setting near-optimal ad allocations can be computed based on structural properties of the input graph if bidders issue individual, equally valued bids. Recall that, without degree bounds, the competitive ratio for the b -matching problem tends from below to $1 - 1/e \approx 0.63$. The competitiveness of c^* improves upon the previous best ratio of $1 - (1 - 1/d)^k$ [19]. The ratio c^* is equal to $1 - (1 - 1/d)^k$ for $b = 1$ and strictly increasing in b , for any $k \geq d$. Our asymptotic competitiveness of 1 is a significant improvement over $1 - (1 - 1/d)^k \approx 1 - 1/e^{k/d}$, for the interesting range of small $k/d \geq 1$. For $k < d$, $1 - (1 - 1/d)^k$ and c^* can become small. The algorithms are still $\frac{1}{2}$ -competitive since they match requests whenever possible. We are aware of only two other online matching problems that admit competitive ratios arbitrarily close to 1. As mentioned above, a randomized algorithm is known for online matching in d -regular unit-capacity graphs [5]. For the general AdWords problem, respective algorithms exist if the input R is generated according to probability distributions [8, 9].

2 An optimal online algorithm

In this section we study the setting that all servers have a uniform capacity of b . We develop our algorithm `WEIGHTEDASSIGNMENT`. While serving requests, the algorithm maintains a value $V(l_s, \delta_s)$, for each server s with load l_s and current degree δ_s . At any point in time during the execution of the online algorithm, the load of a server s denotes the amount of matched edges incident to s , while the current degree indicates the total number of edges incident to s . In order to construct the function V and for the purpose of analysis, we formulate `WEIGHTEDASSIGNMENT` as a primal-dual algorithm. The primal and dual linear programs of the b -matching problem are given below. The primal variables $m(s, r)$ indicate if an edge $\{s, r\} \in E$ belongs to the matching. We have dual variables $x(s)$ and $y(r)$.

In the pseudocode of `WEIGHTEDASSIGNMENT`, also stated below, Line 7 is the actual matching step. A new request r is assigned to a neighboring server s for which the difference $V(l_s, \delta_s + 1) - V(l_s, \delta_s)$ is maximized. $N(r)$ is the set of adjacent servers with remaining capacity. All other instructions essentially update primal and dual variables so that a primal and a dual solution are constructed in parallel.

Observe that no dual variable $y(r)$ of any request r is ever increased by `WEIGHTEDASSIGNMENT`. The dual variable $x(s)$ of a server s can be increased in Lines 9 and 11. It is increased if s is matched to a neighboring request r and, importantly, $x(s)$ is also increased if this r is assigned to a different server.

$$\begin{array}{ll}
 \mathbf{P:} \max & \sum_{\{s,r\} \in E} m(s,r) \\
 \text{s.t.} & \sum_{r:\{s,r\} \in E} m(s,r) \leq b, \quad (\forall s \in S) \\
 & \sum_{s:\{s,r\} \in E} m(s,r) \leq 1, \quad (\forall r \in R) \\
 & m(s,r) \geq 0, \quad (\forall \{s,r\} \in E) \\
 \mathbf{D:} \min & \sum_{s \in S} b \cdot x(s) + \sum_{r \in R} y(r) \\
 \text{s.t.} & x(s) + y(r) \geq 1, \quad (\forall \{s,r\} \in E) \\
 & x(s), y(r) \geq 0, \quad (\forall s \in S, \forall r \in R)
 \end{array}$$

Algorithm 1 `WEIGHTEDASSIGNMENT`.

```

1 Initialize  $x(s) = 0, y(r) = 0$  and  $m(s, r) = 0, \forall s \in S$  and  $\forall r \in R$ ;
2 while a new request  $r \in R$  arrives do
3   | Let  $N(r)$  denote the set of neighbors  $s$  of  $r$  with remaining capacity;
4   | if  $N(r) = \emptyset$  then
5   |   | Do not match  $r$ ;
6   | else
7   |   | Match  $r$  to  $\arg \max \{V(l_s, \delta_s + 1) - V(l_s, \delta_s) : s \in N(r)\}$ ;
8   |   | Update  $m(s, r) \leftarrow 1$ ;
9   |   | Set  $x(s) \leftarrow V(l_s + 1, \delta_s + 1)$ ;
10  |   | forall  $s' \neq s \in N(r)$  do
11  |   |   | Set  $x(s') \leftarrow V(l_{s'}, \delta_{s'} + 1)$ ;
12  |   | end
13  | end
14 end

```

In the analysis, we will see how the function V has to be defined so that `WEIGHTEDASSIGNMENT` achieves the desired competitive ratio c^* . Note that we always construct a feasible dual solution if $x(s) = 1$ holds, for all servers $s \in S$, by the end of the algorithm.

4:6 Tight Bounds for Online b -Matching in Bounded-Degree Graphs

Here lies a crucial idea of the algorithm and the construction of V . We demand $V(b, \delta) = 1$, for all $\delta \geq b$, and $V(l, \delta) = 1$ if $\delta \geq kb$, for all $0 \leq l \leq b$. Also, $V(0, 0) = 0$. These constraints have two important implications.

1. The dual variable $x(s)$ of a server s with load l_s and degree δ_s is always equal to $V(l_s, \delta_s)$: Consider an incoming request r that is a neighbor of s . While $l_s < b$, it holds $N(r) \neq \emptyset$ and r is matched to some server. Lines 9 and 11 correctly update $x(s)$ with respect to the new load and degree values. If $l_s = b$, then inductively by the first constraint $x(s) = 1$ and no update is necessary.
2. The constructed dual solution is feasible: Implication 1 and the second constraint ensure that $x(s) = 1$ holds for all $s \in S$ by the end of the algorithm, since every server s has a degree of at least kb .

Let P and D denote the value of the primal and dual solution constructed by the algorithm, respectively. We denote a change in the value of the primal and dual solution by ΔP and ΔD , respectively. It holds that the size of the matching constructed by `WEIGHTEDASSIGNMENT` is exactly $|\text{ALG}| = P$. Moreover, by weak duality, we get that the size of the optimum matching is $|\text{OPT}| \leq D$. Hence, if we were able to bound $\Delta D \leq \Delta P/c$ at every step, we would obtain a competitive ratio of c .

$$\frac{|\text{ALG}|}{|\text{OPT}|} \geq \frac{P}{D} \geq \frac{P}{\frac{1}{c} \cdot P} = c.$$

Recall that the value of the dual solution is only increased if a request r is matched to a server s . Then, the value of the primal solution is increased by 1, while the value of the dual solution is increased by

$$\Delta D = b \cdot \left(V(l_s + 1, \delta_s + 1) - V(l_s, \delta_s) + \sum_{s' \in N(r) \setminus \{s\}} V(l_{s'}, \delta_{s'} + 1) - V(l_{s'}, \delta_{s'}) \right).$$

The algorithm chooses s such that $V(l_s, \delta_s + 1) - V(l_s, \delta_s) \geq V(l_{s'}, \delta_{s'} + 1) - V(l_{s'}, \delta_{s'})$ holds for all $s' \in N(r)$. Furthermore, $|N(r)| \leq d$ implies that we can bound this increase by

$$\Delta D \leq b \cdot \left(V(l_s + 1, \delta_s + 1) - V(l_s, \delta_s) + (d - 1) \cdot (V(l_s, \delta_s + 1) - V(l_s, \delta_s)) \right).$$

This means, that we need to determine the biggest possible constant $c^* \in (0, 1]$ such that

$$b \cdot \left(V(l + 1, \delta + 1) - V(l, \delta) + (d - 1) \cdot (V(l, \delta + 1) - V(l, \delta)) \right) \leq \frac{1}{c^*} \quad (1)$$

holds for all $0 \leq l < b$ and all $\delta \geq l$, while still satisfying our constraints that $V(b, \cdot) = 1$ and $V(\cdot, \delta') = 1$ for $\delta' \geq kb$. For this, we define

$$p(l, \delta) := V(l + 1, \delta + 1) - V(l, \delta) \quad \text{and} \quad q(l, \delta) := V(l, \delta + 1) - V(l, \delta).$$

In other words, the dual variable $x(s)$ of a server s with load l and current degree δ is increased by $p(l, \delta)$, when a request is assigned to s , and increased by $q(l, \delta)$, when a neighboring request is assigned to a different server. Our constraints immediately give $p(l, \delta) = q(l, \delta) = 0$, if $l = b$ or $\delta \geq kb$. Hence, we will focus on the case $0 \leq l < b$ and $l \leq \delta < kb$ in the following. Rewriting and rearranging inequality (1) in terms of p and q yields

$$q(l, \delta) \leq \frac{1}{d-1} \left(\frac{1}{b \cdot c} - p(l, \delta) \right).$$

We treat the values $p(i, i)$, for $0 \leq i < b - 1$, as the variables of our optimization, since every other p and q value can then be computed based on these choices. To get comfortable with the recursions and ideas in the latter part of this section, we do the following warm-up, where we consider $V(b - 1, \delta)$. It holds

$$V(b - 1, \delta) = \sum_{i=0}^{b-2} p(i, i) + \sum_{j=b-1}^{\delta-1} q(b - 1, j).$$

Our first constraint $V(b, \delta) = 1$, for all $\delta \geq b$, implies that $p(b - 1, \delta) = 1 - V(b - 1, \delta)$. We do not want to waste any potential increases in our dual variables, since we want to maximize c . Thus, we will choose the maximum possible value for $q(b - 1, \delta)$, which is

$$q(b - 1, \delta) = \frac{1}{d - 1} \left(\frac{1}{b \cdot c} - p(b - 1, \delta) \right) = \frac{1}{d - 1} \left(\frac{1}{b \cdot c} - 1 + V(b - 1, \delta) \right).$$

It follows that

$$V(b - 1, \delta + 1) = V(b - 1, \delta) + q(b - 1, \delta) = \frac{d}{d - 1} V(b - 1, \delta) + \frac{1}{d - 1} \left(\frac{1}{b \cdot c} - 1 \right), \quad (2)$$

for all $b - 1 \leq \delta < kb$ and with $V(b - 1, b - 1) = \sum_{i=0}^{b-2} p(i, i)$. To ease notation in the future, we further define $P_i := \sum_{j=0}^{i-1} p(j, j)$, for all $0 \leq i \leq b - 1$, so that we get $P_i = V(i, i)$. Note that $P_0 = 0$. Solving the recurrence relation (2) yields

$$\begin{aligned} V(b - 1, \delta) &= \left(\frac{d}{d - 1} \right)^{\delta - (b-1)} P_{b-1} + \frac{1}{d - 1} \left(\frac{1}{b \cdot c} - 1 \right) \cdot \sum_{i=0}^{\delta - (b-1) - 1} \left(\frac{d}{d - 1} \right)^i \\ &= \left(\frac{d}{d - 1} \right)^{\delta - (b-1)} P_{b-1} + \frac{1}{d - 1} \left(\frac{1}{b \cdot c} - 1 \right) \cdot \frac{\left(\frac{d}{d - 1} \right)^{\delta - (b-1)} - 1}{\left(\frac{d}{d - 1} \right) - 1} \\ &= \left(\frac{d}{d - 1} \right)^{\delta - (b-1)} \left(P_{b-1} + \frac{1}{b \cdot c} - 1 \right) + 1 - \frac{1}{b \cdot c}. \end{aligned}$$

The following lemma generalizes the computation above to all other load levels.

► **Lemma 1.** *For all l , $0 \leq l \leq b$, and for all δ , $l \leq \delta \leq kb$, it holds that*

$$V(l, \delta) = \sum_{i=l}^{b-1} (-1)^{i-l} \frac{1}{(d - 1)^{i-l}} \binom{\delta - l}{i - l} \left(\frac{d}{d - 1} \right)^{\delta - i} \left(P_i + \frac{b - i}{b \cdot c} - 1 \right) + 1 - \frac{b - l}{b \cdot c}. \quad (3)$$

Proof. By induction over l , starting with $l = b$ and going down to $l = 0$. The induction base $l = b$ is true, because we have $V(b, \delta) = 1$. Thus, we focus on the induction step $l + 1 \rightsquigarrow l$. Similar arguments as before yield for $l \leq \delta < kb$

$$q(l, \delta) = \frac{1}{d - 1} \left(\frac{1}{b \cdot c} - p(l, \delta) \right) = \frac{1}{d - 1} \left(\frac{1}{b \cdot c} - V(l + 1, \delta + 1) + V(l, \delta) \right).$$

We can now define the recurrence relation for $V(l, \delta)$

$$V(l, \delta + 1) = V(l, \delta) + q(l, \delta) = \frac{d}{d - 1} V(l, \delta) + \frac{1}{d - 1} \left(\frac{1}{b \cdot c} - V(l + 1, \delta + 1) \right),$$

4:8 Tight Bounds for Online b-Matching in Bounded-Degree Graphs

with $V(l, l) = P_l$. Solving this recurrence yields

$$\begin{aligned} V(l, \delta) &= \left(\frac{d}{d-1}\right)^{\delta-l} P_l + \frac{1}{d-1} \frac{1}{b \cdot c} \sum_{i=0}^{\delta-l-1} \left(\frac{d}{d-1}\right)^i \\ &\quad - \frac{1}{d-1} \sum_{i=0}^{\delta-l-1} \left(\frac{d}{d-1}\right)^i V(l+1, \delta-i). \end{aligned} \tag{4}$$

In the next step, we will need the following fact [11].

► **Fact 2.** For $n, k \in \mathbb{N}_0$, it holds that

$$\sum_{i=0}^n \binom{i}{k} = \binom{n+1}{k+1}.$$

Next, we apply the induction hypothesis to determine $V(l, \delta)$. For clarity, we focus on the last sum of equality (4) first

$$\begin{aligned} &\sum_{i=0}^{\delta-l-1} \left(\frac{d}{d-1}\right)^i V(l+1, \delta-i) \stackrel{\text{IH}}{=} \sum_{i=0}^{\delta-l-1} \left(\frac{d}{d-1}\right)^i \left[\sum_{j=l+1}^{b-1} (-1)^{j-(l+1)} \frac{1}{(d-1)^{j-(l+1)}} \right. \\ &\quad \cdot \left. \binom{\delta-i-(l+1)}{j-(l+1)} \left(\frac{d}{d-1}\right)^{\delta-i-j} \left(P_j + \frac{b-j}{b \cdot c} - 1\right) + 1 - \frac{b-(l+1)}{b \cdot c} \right] \\ &= \sum_{j=l+1}^{b-1} \left[(-1)^{j-(l+1)} \frac{1}{(d-1)^{j-(l+1)}} \left(\frac{d}{d-1}\right)^{\delta-j} \left(P_j + \frac{b-j}{b \cdot c} - 1\right) \right. \\ &\quad \cdot \left. \sum_{i=0}^{\delta-l-1} \binom{\delta-i-(l+1)}{j-(l+1)} \right] + \left(1 - \frac{b-(l+1)}{b \cdot c}\right) \sum_{i=0}^{\delta-l-1} \left(\frac{d}{d-1}\right)^i \\ &= \sum_{j=l+1}^{b-1} (-1)^{j-(l+1)} \frac{1}{(d-1)^{j-(l+1)}} \left(\frac{d}{d-1}\right)^{\delta-j} \left(P_j + \frac{b-j}{b \cdot c} - 1\right) \sum_{i=0}^{\delta-l-1} \binom{i}{j-(l+1)} \\ &\quad + \left(1 - \frac{b-(l+1)}{b \cdot c}\right) (d-1) \left(\left(\frac{d}{d-1}\right)^{\delta-l} - 1 \right) \\ &= \sum_{j=l+1}^{b-1} (-1)^{j-(l+1)} \frac{1}{(d-1)^{j-(l+1)}} \left(\frac{d}{d-1}\right)^{\delta-j} \left(P_j + \frac{b-j}{b \cdot c} - 1\right) \binom{\delta-l}{j-l} \\ &\quad + \left(1 - \frac{b-(l+1)}{b \cdot c}\right) (d-1) \left(\left(\frac{d}{d-1}\right)^{\delta-l} - 1 \right), \end{aligned}$$

where we used Fact 2 in the last step. Now, we can finish the induction step by plugging this into (4)

$$\begin{aligned}
V(l, \delta) &= \left(\frac{d}{d-1}\right)^{\delta-l} P_l + \frac{1}{b \cdot c} \left(\left(\frac{d}{d-1}\right)^{\delta-l} - 1 \right) \\
&\quad + \sum_{j=l+1}^{b-1} (-1)^{j-l} \frac{1}{(d-1)^{j-l}} \left(\frac{d}{d-1}\right)^{\delta-j} \left(P_j + \frac{b-j}{b \cdot c} - 1 \right) \binom{\delta-l}{j-l} \\
&\quad - \left(1 - \frac{b-(l+1)}{b \cdot c} \right) \left(\left(\frac{d}{d-1}\right)^{\delta-l} - 1 \right) \\
&= \left(\frac{d}{d-1}\right)^{\delta-l} \left(P_l + \frac{1}{b \cdot c} + \frac{b-(l+1)}{b \cdot c} - 1 \right) + 1 - \frac{1}{b \cdot c} - \frac{b-(l+1)}{b \cdot c} \\
&\quad + \sum_{j=l+1}^{b-1} (-1)^{j-l} \frac{1}{(d-1)^{j-l}} \left(\frac{d}{d-1}\right)^{\delta-j} \left(P_j + \frac{b-j}{b \cdot c} - 1 \right) \binom{\delta-l}{j-l} \\
&= \sum_{j=l}^{b-1} (-1)^{j-l} \frac{1}{(d-1)^{j-l}} \left(\frac{d}{d-1}\right)^{\delta-j} \left(P_j + \frac{b-j}{b \cdot c} - 1 \right) \binom{\delta-l}{j-l} + 1 - \frac{b-l}{b \cdot c}. \quad \blacktriangleleft
\end{aligned}$$

So far, we have only leveraged our constraint $V(b, \cdot) = 1$. With our description of $V(l, \delta)$, for all $0 \leq l \leq b$ and $l \leq \delta \leq kb$, we can also leverage $V(\cdot, kb) = 1$ to determine P_i , for all $0 \leq i \leq b-1$. For this, we will need the following technical lemma. The proof is given in the full version of the paper.

► **Lemma 3.** For $k, n, m \in \mathbb{N}$, with $m \geq n \geq k$, it holds that

$$\sum_{i=1}^{n-k} (-1)^i \binom{m}{i} \binom{m-i}{n-k-i} = -\binom{m}{n-k}.$$

► **Lemma 4.** For all l , $0 \leq l \leq b-1$, it holds that

$$\left(\frac{d}{d-1}\right)^{kb-l} \left(P_l + \frac{b-l}{b \cdot c} - 1 \right) = \frac{1}{b \cdot c} \left(\sum_{i=1}^{b-l} i \binom{kb-l}{b-l-i} \frac{1}{(d-1)^{b-l-i}} \right). \quad (5)$$

Proof. By induction over l from $l = b-1$ down to $l = 0$. We start with the induction base $l = b-1$. Our second constrain yields $V(b-1, kb) = 1$. It then follows from Lemma 1 that

$$V(b-1, kb) = \left(\frac{d}{d-1}\right)^{kb-(b-1)} \left(P_{b-1} + \frac{1}{b \cdot c} - 1 \right) + 1 - \frac{1}{b \cdot c} \stackrel{\text{def.}}{=} 1,$$

which immediately finishes the induction base, since the right-hand side of (5) is simply $1/(b \cdot c)$.

We can now move on to the induction step $\forall i > l \rightsquigarrow l$. We use $V(l, kb) = 1$ and rearrange with the help of Lemma 1

$$\begin{aligned}
\left(\frac{d}{d-1}\right)^{kb-l} \left(P_l + \frac{b-l}{b \cdot c} - 1 \right) &= \frac{b-l}{b \cdot c} \\
- \sum_{i=l+1}^{b-1} (-1)^{i-l} \frac{1}{(d-1)^{i-l}} \binom{kb-l}{i-l} \left(\frac{d}{d-1}\right)^{kb-i} \left(P_i + \frac{b-i}{b \cdot c} - 1 \right) &. \quad (6)
\end{aligned}$$

4:10 Tight Bounds for Online b -Matching in Bounded-Degree Graphs

It is now possible to apply the induction hypothesis. For clarity, we focus on the second line of (6)

$$\begin{aligned} & \sum_{i=l+1}^{b-1} (-1)^{i-l} \frac{1}{(d-1)^{i-l}} \binom{kb-l}{i-l} \frac{1}{b \cdot c} \left(\sum_{j=1}^{b-i} j \binom{kb-i}{b-i-j} \frac{1}{(d-1)^{b-i-j}} \right) \\ &= \frac{1}{b \cdot c} \left(\sum_{a=1}^{b-l-1} (-1)^a \sum_{j=1}^{b-(l+a)} j \frac{1}{(d-1)^{b-l-j}} \binom{kb-l}{a} \binom{kb-(l+a)}{b-(l+a)-j} \right), \end{aligned}$$

where we substituted $a := i - l$. Next, we carefully swap these nested sums. For this, observe that, for a fixed value j , we have exactly $b - l - j$ addends, more precisely, one addend for each $1 \leq a \leq b - l - j$

$$\begin{aligned} & \frac{1}{b \cdot c} \left(\sum_{a=1}^{b-l-1} (-1)^a \sum_{j=1}^{b-(l+a)} j \frac{1}{(d-1)^{b-l-j}} \binom{kb-l}{a} \binom{kb-l-a}{b-l-a-j} \right) \\ &= \frac{1}{b \cdot c} \left(\sum_{j=1}^{b-l-1} j \frac{1}{(d-1)^{b-l-j}} \sum_{a=1}^{b-l-j} (-1)^a \binom{kb-l}{a} \binom{kb-l-a}{b-l-a-j} \right) \\ &= -\frac{1}{b \cdot c} \left(\sum_{j=1}^{b-l-1} j \frac{1}{(d-1)^{b-l-j}} \binom{kb-l}{b-l-j} \right), \end{aligned}$$

where we applied Lemma 3 with $k = j$, $n = b - l$ and $m = kb - l$ in the last step. Plugging this back into (6) gives

$$\begin{aligned} \left(\frac{d}{d-1} \right)^{kb-l} \left(P_l + \frac{b-l}{b \cdot c} - 1 \right) &= \frac{b-l}{b \cdot c} + \frac{1}{b \cdot c} \left(\sum_{j=1}^{b-l-1} j \frac{1}{(d-1)^{b-l-j}} \binom{kb-l}{b-l-j} \right) \\ &= \frac{1}{b \cdot c} \left(\sum_{j=1}^{b-l} j \frac{1}{(d-1)^{b-l-j}} \binom{kb-l}{b-l-j} \right). \quad \blacktriangleleft \end{aligned}$$

With the help of Lemma 4, we can finally determine the resulting constant c^* . For $l = 0$, we have $P_0 = 0$, and thus

$$\left(\frac{d}{d-1} \right)^{kb} \left(\frac{1}{c^*} - 1 \right) = \frac{1}{b \cdot c^*} \left(\sum_{i=1}^b i \binom{kb}{b-i} \frac{1}{(d-1)^{b-i}} \right),$$

where solving for c^* yields

$$c^* = 1 - \frac{1}{b} \left(\sum_{i=1}^b i \binom{kb}{b-i} \frac{1}{(d-1)^{b-i}} \right) \left(1 - \frac{1}{d} \right)^{kb}.$$

► **Theorem 5.** *WEIGHTEDASSIGNMENT achieves a competitive ratio of c^* for the online b -matching problem with uniform server capacities on (k, d) -graphs.*

Lemma 1, together with Lemma 4 and c^* , specifies the function V . Its values can be calculated in a preprocessing step and accessed by table lookup when WEIGHTEDASSIGNMENT serves requests. The parameters k and d must be known. In an application, they can be learned over time. Alternatively, one can work with conservative estimates. Figure 1 shows

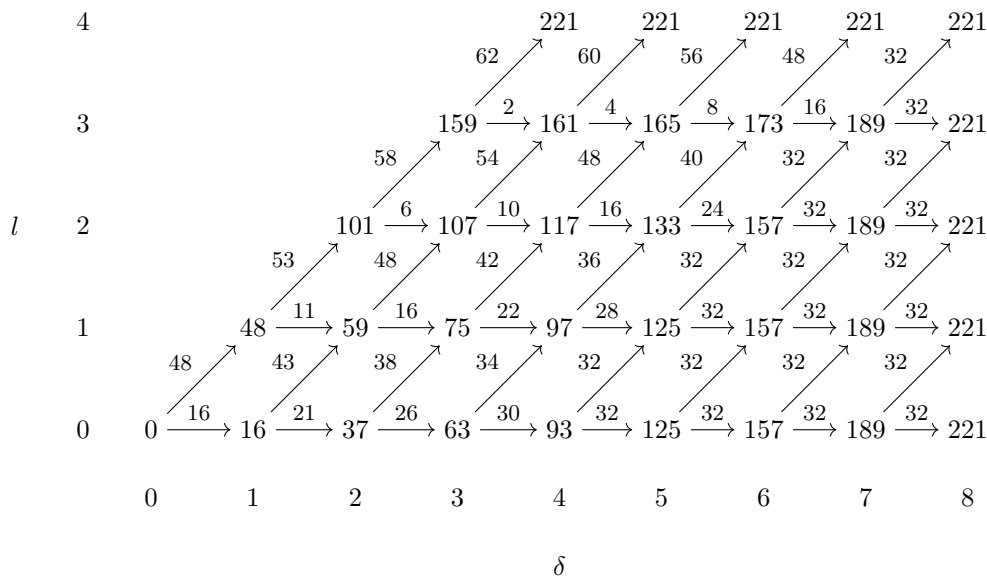


Figure 1 The function V for $k = d = 2$ and $b = 4$. All values are multiplied by 221.

the function V for a small example with $k = d = 2$ and $b = 4$. In this case, we have $c^* = 221/256$ and $1/(bc^*) = 64/221$. The arrows depict the possible increases in V for every l , $0 \leq l < b$, and δ , $l \leq \delta < kb$, i.e. horizontal arrows denote the $q(l, \delta)$ values and diagonal arrows the $p(l, \delta)$ values. All actual values are multiplied by 221 in order to eliminate fractions. Observe that $p(l, \delta) + (d - 1)q(l, \delta) = 1/(bc^*)$ holds for all l and δ . This is what allows us to bound the total increase in the dual solution by $1/c^*$, if we always pick the neighboring server s that maximizes $q(l_s, \delta_s)$. For the matching decisions, WEIGHTEDASSIGNMENT only uses the horizontal arrows. Notice that WEIGHTEDASSIGNMENT is different from a BALANCE algorithm that breaks ties by HIGHDEGREE, or from a HIGHDEGREE algorithm that breaks ties by BALANCE. For example, WEIGHTEDASSIGNMENT prefers a server with load 1 and degree 5 to a server with load 0 and degree 1, whom it then prefers to a server with load 3 and degree 6.

3 Upper bounds

We will show that WEIGHTEDASSIGNMENT is optimal for (k, d) -graphs with $k \geq d$, i.e. no deterministic online algorithm can achieve a competitive ratio better than c^* . We start by proving this for the online b -matching problem with uniform server capacities, and later extend it to the more general problems.

First, we show that any (k, d) -graph with uniform server capacities b has a perfect b -matching, i.e. a matching where every server $s \in S$ is matched exactly b times, if $k \geq d$. This generalizes Lemma 6.1 in [19], which states this for $b = 1$. The proof is given in the full version of this paper.

► **Lemma 6.** *Every (k, d) -graph $G = (S \cup R, E)$, where $k \geq d$, with uniform server capacities b has a perfect b -matching.*

We move on to describing the adversary input. We start by following the construction of the previously known upper bound detailed in [19]. There are $N = d^{kb}$ servers, and the requests arrive in kb rounds. Let S_i denote the set of unmatched servers at the beginning

4:12 Tight Bounds for Online b -Matching in Bounded-Degree Graphs

of round i , $0 \leq i < kb$. It will hold that every server in S_i has a current degree of i and that $|S_i| = N(1 - 1/d)^i$. Note that this number is always a multiple of d , by choice of N . During round i , $|S_i|/d$ requests are introduced, such that every request is adjacent to exactly d distinct servers in S_i and every server in S_i gains one new neighboring request. If the online algorithm decides not to match an introduced request, we consider it matched to an arbitrary neighbor. This will only improve the performance of said algorithm. This means that a $(1 - 1/d)$ fraction of the servers in S_i are still unmatched after round i , explaining the previously mentioned $|S_i| = N(1 - 1/d)^i$. Thus, there are $N \cdot (1 - 1/d)^{kb}$ servers with degree kb and load 0 after round $kb - 1$, irrespective of what choices the algorithm makes. In a final round, further requests are introduced to all matched servers arbitrarily, such that we get a valid (k, d) -graph. The online algorithm will have matched at most $bN \cdot (1 - (1 - 1/d)^{kb})$ requests, while Lemma 6 implies that an optimal offline algorithm matches bN requests. This yields the previously known upper bound of $(1 - (1 - 1/d)^{kb})$.

However, it seems suboptimal to introduce requests arbitrarily after the initial kb rounds. In fact, we will show that we can further limit the number of requests matched by the online algorithm if we introduce the requests more carefully. Note that all the servers that are matched during round i of the previous input are similar in the sense that they all have degree i and load 1. We will apply the ideas above recursively to the sets of matched servers of all rounds. More precisely, let T denote the set of matched server during some round. Say all servers in T have load l , $0 \leq l < b$, and degree δ , $l \leq \delta < kb$. We then schedule $kb - \delta$ rounds for T using the same construction as above. Let T_j denote the set of servers that still have load l at the beginning of round j , $0 \leq j < kb - \delta$. It will now hold that every server in T_j has a current degree of $\delta + j$ and that $|T_j| = |T|(1 - 1/d)^j$. We have to make sure that all the possible values of $|T_j|$ are multiples of d . This is done by increasing the initial number of servers N adequately. After these $kb - \delta$ round, we have $|T| \cdot (1 - 1/d)^{kb - \delta}$ servers with degree kb and load l . This process is repeated for all the sets of matched servers until we eventually obtain a valid (k, d) -graph, in which every server has degree kb .

For this, we formally define a function F , where $F(x, l, \delta)$ denotes how many units of capacity we can force a deterministic online algorithm to leave empty when starting with x servers that all have load l and degree δ . This allows us to upper bound the number of matched requests by $bN - F(N, 0, 0)$, yielding the following upper bound

$$c \leq \frac{bN - F(N, 0, 0)}{bN} = 1 - \frac{F(N, 0, 0)}{bN}. \quad (7)$$

We cannot create any empty spots on full servers, so we have $F(x, b, \delta) = 0$, for all $b \leq \delta \leq kb$. Once a server has kb adjacent requests, we have satisfied the (k, d) -graph property locally, so we do not need to introduce any more adjacent requests for this server. This is captured by $F(x, l, kb) = x \cdot (b - l)$, for all $0 \leq l \leq b$. For all other combinations of l and δ , it is possible to define F recursively. Recall that we introduce $kb - \delta$ rounds when starting with x servers all with load l and degree δ . During each of these rounds, exactly a $1/d$ fraction of the servers that still had load l at the start of the round are discarded. Moreover, every server gets exactly one new neighbor during each round until they are matched. This implies

$$F(x, l, \delta) = x \cdot \left(1 - \frac{1}{d}\right)^{kb - \delta} (b - l) + \sum_{i=1}^{kb - \delta} F\left(x \cdot \frac{1}{d} \left(1 - \frac{1}{d}\right)^{i-1}, l + 1, \delta + i\right), \quad (8)$$

for all $0 \leq l < b$ and $l \leq \delta < kb$. The following lemma solves this recurrence, which we can then apply in (7) to obtain the theorem.

► **Lemma 7.** For all l , $0 \leq l \leq b$, and all δ , $l \leq \delta \leq kb$, it holds that

$$F(x, l, \delta) = x \left(1 - \frac{1}{d}\right)^{kb-\delta} \left(\sum_{i=1}^{b-l} i \binom{kb-\delta}{b-l-i} \frac{1}{(d-1)^{b-l-i}} \right). \quad (9)$$

Proof. By induction over l , starting with $l = b$ and going down to $l = 0$. The induction base is satisfied as we get the empty sum in (9), making the whole expression zero. In the induction step $l+1 \rightsquigarrow l$, we can apply our induction hypothesis to F in (8).

$$\begin{aligned} & \sum_{i=1}^{kb-\delta} F \left(x \cdot \frac{1}{d} \left(1 - \frac{1}{d}\right)^{i-1}, l+1, \delta+i \right) \\ & \stackrel{\text{IH}}{=} \sum_{i=1}^{kb-\delta} x \frac{1}{d} \left(1 - \frac{1}{d}\right)^{i-1} \left(1 - \frac{1}{d}\right)^{kb-(\delta+i)} \left(\sum_{j=1}^{b-(l+1)} j \binom{kb-\delta-i}{b-(l+1)-j} \frac{1}{(d-1)^{b-(l+1)-j}} \right) \\ & = x \frac{1}{d} \left(1 - \frac{1}{d}\right)^{kb-\delta-1} \left(\sum_{j=1}^{b-(l+1)} j \frac{1}{(d-1)^{b-(l+1)-j}} \sum_{i=1}^{kb-\delta} \binom{kb-\delta-i}{b-(l+1)-j} \right) \\ & = x \left(1 - \frac{1}{d}\right)^{kb-\delta} \frac{1}{d-1} \left(\sum_{j=1}^{b-(l+1)} j \frac{1}{(d-1)^{b-l-j-1}} \sum_{i=0}^{kb-\delta-1} \binom{i}{b-l-j-1} \right) \\ & = x \left(1 - \frac{1}{d}\right)^{kb-\delta} \left(\sum_{j=1}^{b-(l+1)} j \frac{1}{(d-1)^{b-l-j}} \binom{kb-\delta}{b-l-j} \right), \end{aligned}$$

where we used Fact 2 in the last step. Finally, we can plug this result back into (8) to obtain

$$\begin{aligned} F(x, l, \delta) & = x \cdot \left(1 - \frac{1}{d}\right)^{kb-\delta} (b-l) + \sum_{i=1}^{kb-\delta} F \left(x \cdot \frac{1}{d} \left(1 - \frac{1}{d}\right)^{i-1}, l+1, \delta+i \right) \\ & = x \cdot \left(1 - \frac{1}{d}\right)^{kb-\delta} \left((b-l) + \sum_{j=1}^{b-(l+1)} j \frac{1}{(d-1)^{b-l-j}} \binom{kb-\delta}{b-l-j} \right) \\ & = x \cdot \left(1 - \frac{1}{d}\right)^{kb-\delta} \left(\sum_{j=1}^{b-l} j \frac{1}{(d-1)^{b-l-j}} \binom{kb-\delta}{b-l-j} \right). \quad \blacktriangleleft \end{aligned}$$

► **Theorem 8.** No deterministic online algorithm for the b -matching problem with uniform server capacities b can achieve a competitiveness better than c^* on (k, d) -graphs with $k \geq d$.

We extend this upper bound to the more general case with variable server capacities, which we will examine in the next section. Let $b_{\min} = \min_{s \in S} b_s$. The optimal competitiveness is then

$$c_{\min}^* = 1 - \frac{1}{b_{\min}} \left(\sum_{i=1}^{b_{\min}} i \binom{kb_{\min}}{b_{\min}-i} \frac{1}{(d-1)^{b_{\min}-i}} \right) \left(1 - \frac{1}{d}\right)^{kb_{\min}}.$$

► **Corollary 9.** No deterministic online algorithm for the b -matching problem with variable server capacities can achieve a competitive ratio better than c_{\min}^* on (k, d) -graphs with $k \geq d$.

The proof is detailed in the full version of this paper. Obviously, Theorem 8 and Corollary 9 also hold for the more general vertex-weighted b -matching problem, addressed in the next section.

4 Variable server capacities and vertex weights

We detail the necessary changes to `WEIGHTEDASSIGNMENT` such that it can handle variable server capacities as well as vertex weights, while still achieving the optimal competitive ratio.

4.1 Variable server capacities

Recall that every server $s \in S$ now has a server capacity b_s , and thus a degree of at least $d(s) \geq k \cdot b_s$. This changes the objective function of the dual linear program to

$$\sum_{s \in S} b_s \cdot x(s) + \sum_{r \in R} y(r).$$

We handle this by computing the function V_s for every server individually, for its capacity b_s . This means that we construct V_s according to Section 2, such that

$$b_s \cdot \left(V_s(l+1, \delta+1) - V_s(l, \delta) + (d-1) \cdot (V_s(l, \delta+1) - V_s(l, \delta)) \right) \leq \frac{1}{c_s^*}, \quad (10)$$

where c_s^* is equal to c^* , but b is replaced by b_s . Moreover, we have $V_s(b_s, \cdot) = 1$ and $V_s(\cdot, \delta') = 1$ if $\delta' \geq kb_s$, meaning that we again construct a feasible dual solution. However, we still have to adapt the decision criterion of `WEIGHTEDASSIGNMENT`. We change Line 7 in Algorithm 1 to

$$\text{Match } r \text{ to } \arg \max \{ b_s \cdot (V_s(l_s, \delta_s + 1) - V_s(l_s, \delta_s)) : s \in N(r) \}.$$

This allows us to upper bound the total increase in the dual solution when the adapted strategy, called `WEIGHTEDASSIGNMENT(VC)`, assigns a request r to a server s by

$$\begin{aligned} \Delta D &= b_s \cdot \left(V_s(l_s + 1, \delta_s + 1) - V_s(l_s, \delta_s) \right) \\ &\quad + \sum_{s' \in N(r) \setminus \{s\}} b_{s'} \cdot \left(V_{s'}(l_{s'}, \delta_{s'} + 1) - V_{s'}(l_{s'}, \delta_{s'}) \right) \\ &\leq b_s \cdot \left(V_s(l_s + 1, \delta_s + 1) - V_s(l_s, \delta_s) + (d-1) \cdot (V_s(l_s, \delta_s + 1) - V_s(l_s, \delta_s)) \right) \leq \frac{1}{c_s^*}. \end{aligned}$$

Thus, `WEIGHTEDASSIGNMENT(VC)` achieves a competitive ratio of $\min_{s \in S} c_s^*$. In Section 5 we will show that c^* is monotonically increasing in b for $k \geq d$, meaning that $\min_{s \in S} c_s^* = c_{\min}^*$, cf. Section 3.

► **Theorem 10.** *`WEIGHTEDASSIGNMENT(VC)` achieves a competitive ratio of $\min_{s \in S} c_s^*$ the b -matching problem with variable server capacities on (k, d) -graphs. The ratio equals c_{\min}^* and is optimal for $k \geq d$.*

4.2 Vertex weights

At last, we consider the vertex-weighted extension of the online b -matching problem. Every server $s \in S$ now has a weight w_s assigned to it, and the value of every matching edge incident to s is multiplied by w_s . This problem is modelled by the following linear programs.

$$\begin{aligned}
\text{Primal: } \max \quad & \sum_{\{s,r\} \in E} w_s \cdot m(s,r) \\
\text{s.t.} \quad & \sum_{r:\{s,r\} \in E} w_s \cdot m(s,r) \leq w_s \cdot b_s, & (\forall s \in S) \\
& \sum_{s:\{s,r\} \in E} m(s,r) \leq 1, & (\forall r \in R) \\
& m(s,r) \geq 0, & (\forall \{s,r\} \in E).
\end{aligned}$$

$$\begin{aligned}
\text{Dual: } \min \quad & \sum_{s \in S} w_s \cdot b_s \cdot x(s) + \sum_{r \in R} y(r) \\
\text{s.t.} \quad & w_s \cdot x(s) + y(r) \geq w_s, & (\forall \{s,r\} \in E) \\
& x(s), y(r) \geq 0, & (\forall s \in S, \forall r \in R).
\end{aligned}$$

Observe that $x(s) = 1$, for all $s \in S$, by the end of the algorithm still implies dual feasibility. Hence, we do not need to change the construction of V_s , and still obtain a feasible dual solution. All we need to do is to change the decision criterion of `WEIGHTEDASSIGNMENT(VC)` once more to

$$\text{Match } r \text{ to } \arg \max \{w_s \cdot b_s \cdot (V_s(l_s, \delta_s + 1) - V_s(l_s, \delta_s)) : s \in N(r)\}.$$

Whenever the resulting algorithm `WEIGHTEDASSIGNMENT(VW)` assigns a request r to a server s , we increase the primal solution by w_s , while we can upper bound the increase in the dual solution by

$$\Delta D \leq w_s b_s \cdot \left(V_s(l_s + 1, \delta_s + 1) - V_s(l_s, \delta_s) + (d-1)(V_s(l_s, \delta_s + 1) - V_s(l_s, \delta_s)) \right) \leq \frac{w_s}{c_s^*}.$$

This again yields a competitiveness of $\min_{s \in S} c_s^*$.

► **Theorem 11.** *`WEIGHTEDASSIGNMENT(VW)` achieves a competitive ratio of $\min_{s \in S} c_s^*$ for the vertex-weighted b -matching problem with variable server capacities on (k, d) -graphs. The ratio equals c_{\min}^* and is optimal for $k \geq d$.*

5 Analysis of the competitive ratio

► **Theorem 12.** *If $k \geq d \geq 2$, the competitive ratio c^* converges to one as b tends to infinity, that is $\lim_{b \rightarrow \infty} c^* = 1$.*

► **Theorem 13.** *If $k \geq d \geq 2$, the competitive ratio c^* is strictly increasing in b , for $b \geq 1$.*

The proofs of these two theorems are given in the full version of the paper. Theorem 12 is shown with the help of Gaussian hypergeometric functions. They allow us to upper bound c^* with a closed expression. The convergence of this expression can then be shown with the help of Stirling's approximation. Since this does not prove monotonicity, we consider the fraction of c^* with $b+1$ over c^* with b in the proof of Theorem 13. We show that this fraction is lower bounded by 1, again with the help of hypergeometric functions.

References

- 1 G. Aggarwal, G. Goel, C. Karande, and A. Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1253–1264. SIAM, 2011.
- 2 Y. Azar, I.R. Cohen, and A. Roytman. Online lower bounds via duality. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1038–1050. SIAM, 2017.
- 3 N. Buchbinder, K. Jain, and J. Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Proc. 15th Annual European Symposium on Algorithms (ESA)*, Springer LNCS, Volume 4698, pages 253–264, 2007.
- 4 K. Chaudhuri, C. Daskalakis, R.D. Kleinberg, and H. Lin. Online bipartite perfect matching with augmentations. In *Proc. 28th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1044–1052, 2009.
- 5 I.R. Cohen and D. Wajc. Randomized online matching in regular graphs. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 960–979. SIAM, 2018.
- 6 R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Comb.*, 21(1):5–12, 2001.
- 7 J. Csima and L. Lovász. A matching algorithm for regular bipartite graphs. *Discret. Appl. Math.*, 35(3):197–203, 1992.
- 8 N.R. Devanur and T.P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proc. 10th ACM Conference on Electronic Commerce (EC)*, pages 71–78. ACM, 2009.
- 9 N.R. Devanur, B. Sivan, and Y. Azar. Asymptotically optimal algorithm for stochastic adwords. In *Proc. 13th ACM Conference on Electronic Commerce (EC)*, pages 388–404. ACM, 2012.
- 10 A. Goel, M. Kapralov, and S. Khanna. Perfect matchings in $O(n \log n)$ time in regular bipartite graphs. *SIAM J. Comput.*, 42(3):1392–1404, 2013.
- 11 R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 1989.
- 12 E.F. Grove, M.-Y. Kao, P. Krishnan, and J.S. Vitter. Online perfect matching and mobile computing. In *Proc. 4th International Workshop on Algorithms and Data Structures (WADS)*, LNCS, Volume 955, pages 194–205. Springer, 1995.
- 13 P. Jaillet and X. Lu. Online stochastic matching: New algorithms with better bounds. *Math. Oper. Res.*, 39(3):624–646, 2014.
- 14 B. Kalyanasundaram and K. Pruhs. An optimal deterministic algorithm for online b-matching. *Theor. Comput. Sci.*, 233(1-2):319–325, 2000.
- 15 R.M. Karp, U.V. Vazirani, and V.V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 352–358, 1990.
- 16 M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: An approach based on strongly factor-revealing LPs. In *Proc. 43rd ACM Symposium on Theory of Computing (STOC)*, pages 597–606, 2011.
- 17 V.H. Manshadi, S. Oveis Gharan, and A. Saberi. Online stochastic matching: Online actions based on offline statistics. *Math. Oper. Res.*, 37(4):559–573, 2012.
- 18 A. Mehta, A. Saberi, U.V. Vazirani, and V.V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.
- 19 J. Naor and D. Wajc. Near-optimum online ad allocation for targeted advertising. *ACM Trans. Economics and Comput.*, 6(3-4):16:1–16:20, 2018.
- 20 A. Schrijver. Bipartite edge coloring in $O(\Delta m)$ time. *SIAM J. Comput.*, 28(3):841–846, 1998.
- 21 V.V. Vazirani. Randomized online algorithms for Adwords. *CoRR*, abs/2107.10777, 2021. [arXiv:2107.10777](https://arxiv.org/abs/2107.10777).

TSP in a Simple Polygon

Henk Alkema ✉

Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

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

Leonidas Theocharous ✉

Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Abstract

We study the Traveling Salesman Problem inside a simple polygon. In this problem, which we call TSP IN A SIMPLE POLYGON, we wish to compute a shortest tour that visits a given set S of n sites inside a simple polygon P with m edges while staying inside the polygon. This natural problem has, to the best of our knowledge, not been studied so far from a theoretical perspective. It can be solved exactly in $\text{poly}(n, m) + 2^{O(\sqrt{n} \log n)}$ time, using an algorithm by Marx, Pilipczuk, and Pilipczuk (FOCS 2018) for SUBSET TSP as a subroutine. We present a much simpler algorithm that solves TSP IN A SIMPLE POLYGON directly and that has the same running time.

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

Keywords and phrases Traveling Salesman Problem, Subexponential algorithms, TSP with obstacles

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.5

Funding HA, MdB, and LT are supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.

1 Introduction

The TRAVELING SALESMAN PROBLEM, or TSP for short, is a classic algorithmic problem. Given an edge-weighted graph, the goal is to compute a *tour* – a cycle that visits each node exactly once – of minimum total weight. The problem has been studied widely; in fact, TSP is probably one of the most intensely studied problems in optimization and computer science. There are even several books devoted to TSP [2, 9, 15, 29]. Important special cases of TSP are METRIC TSP and EUCLIDEAN TSP. In METRIC TSP, the edge weights in the input graph form a metric and, in particular, satisfy the triangle inequality. The algorithm by Christofides [8] gives a $(3/2)$ -approximation for this version of the problem. A special case of METRIC TSP is EUCLIDEAN TSP. Here the input is a set S of n points in \mathbb{R}^d and the goal is to visit all points with a tour of minimum total Euclidean length. Due to its natural setting, EUCLIDEAN TSP is among the most studied versions of TSP. EUCLIDEAN TSP is NP-hard [12, 27] but, unlike METRIC TSP, it admits a PTAS as was shown in the celebrated papers of Arora [3] and (for 2D) Mitchell [26]. A PTAS with a better running time was later presented by Rao and Smith [28]. Recently, Kisfaludi-Bak *et al.* [20] presented a PTAS with $2^{O(1/\varepsilon^{d-1})} n \log n$ running time, which they proved to be optimal under Gap-ETH. There are also PTASs for TSP in planar graphs [4, 21] and in spaces of constant doubling dimension [5] – as Trevisan [33] proved, the restriction to bounded dimension is necessary for a PTAS, even in Euclidean spaces – and in spaces of so-called bounded global growth [7].

Our focus is on exact algorithms for TSP. The general TSP problem can be solved exactly in $O(2^{n^2})$ time using dynamic programming, as was shown independently by Held and Karp [16] and Bellman [6]. There is no subexponential algorithm – that is, no algorithm



© Henk Alkema, Mark de Berg, Morteza Monemizadeh, and Leonidas Theocharous; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 5; pp. 5:1–5:14

Leibniz International Proceedings in Informatics



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

with $2^{o(n)}$ running time – for the general problem, under the Exponential-Time Hypothesis (ETH) [10, Theorem 14.6]. This lower bound even holds for METRIC TSP. On the other hand, EUCLIDEAN TSP can be solved in subexponential time. Already in the early 1990s, Kann [18] and Hwang, Chang and Lee [17] gave $2^{O(\sqrt{n} \log n)}$ algorithms for the planar version of the problem.¹ This was generalized by Smith and Wormald [31], who presented an $2^{O(n^{1-1/d} \log n)}$ algorithm for EUCLIDEAN TSP in \mathbb{R}^d ; here and in the sequel we consider the dimension d to be a fixed constant. For a long time it was open if EUCLIDEAN TSP in \mathbb{R}^d admits an exact algorithm with a running time $2^{O(n^{1-1/d})}$. Recently, the question was settled by De Berg *et al.* [11], who presented such an algorithm and showed that no $2^{o(n^{1-1/d})}$ algorithm exists unless ETH fails. There is also an exact algorithm for TSP in hyperbolic spaces of Gaussian curvature -1 , which runs in quasi-polynomial time if the minimum distance between any two points is at least some fixed constant $\alpha > 0$ [19]. Finally, Klein and Marx [22] presented an algorithm for SUBSET TSP on planar graphs with integer edge weights, where the goal is to compute a shortest tour visiting a given subset of the vertices. Their algorithm runs in time $\text{poly}(|V|) \cdot 2^{O(\sqrt{n} \log n)} + W$, where $|V|$ is the total number of vertices, n is the number of vertices to be visited, and W is the maximum edge weight. This was later improved by Marx, Pilipczuk, and Pilipczuk [24] who presented an algorithm with running time $\text{poly}(|V|) \cdot 2^{O(\sqrt{n} \log n)}$ that does not need the weights to be bounded or integral.

A natural generalization of EUCLIDEAN TSP is to consider a salesman who is moving among a set of obstacles in the plane, or some higher-dimensional space. We call this problem TSP WITH OBSTACLES. As far as we know, and to our surprise, TSP WITH OBSTACLES seems not to have been studied at all from a theoretical perspective, although it has appeared in a behavioural study where subjects were tested on finding the optimal tour [30]. (There is also a paper describing a genetic algorithm for TSP in the presence of obstacles [13], although the setting is slightly different.) A somewhat related problem is where one is given two simple disjoint polygons, and the salesman must visit all vertices of the polygons without crossing the boundary of the polygons. This problem is markedly different from TSP WITH OBSTACLES, however. In particular, it is not a generalization of EUCLIDEAN TSP, and it has been shown by Abrahamson and Shokoufandeh [1] to be solvable in polynomial time.

The fact that TSP WITH OBSTACLES has not been studied is remarkable, since motion-planning and shortest-path problems are among the most widely studied problems in computational geometry. It is beyond the scope of our paper to give an overview of this area, so we just mention one result that we will need. Guibas and Hershberger [14] show how to construct, for a simple polygon P with m vertices and a source point s , a shortest-path map in $O(m)$ time. The shortest-path map allows us to compute, for a point set S of n points in P and a given source point $s \in S$, the shortest-path tree rooted at s and going to all other points in S , in $O(m + n \log m)$ time. If we do this for all points from S as source point, we obtain all shortest paths (and all distances) between the points in S in $O(nm + n^2 \log n)$ time.

TSP WITH OBSTACLES can be solved by computing a shortest path between each pair of points in S , and then running the Bellman-Held-Karp algorithm [6, 16] using the computed pairwise distances, but this leads to an exponential running time. In the plane a much faster algorithm can be obtained if one uses an algorithm for SUBSET TSP as a subroutine, as follows. First, compute all pairwise shortest paths. Next, turn this collection of paths

¹ Comparing the exact lengths of two given tours in the plane is in fact non-trivial, as it involves comparing sums of square roots. To focus on the combinatorial complexity of the problem, such algebraic issues are typically ignored by working in the real-RAM model, which we do as well.

in the plane into a weighted planar graph by inserting a vertex at every intersection point between two paths, where the weight of an edge in the graph is the shortest-path distance between its endpoints. Note that some pairs of path may overlap instead of intersect; in that case, insert a vertex at the two outermost points where they overlap. The resulting graph has $O(n^4)$ vertices – the points in S plus the intersection points – of which n nodes must be visited. Solving SUBSET TSP using the algorithm of Marx, Pilipczuk, and Piliczuk [24] leads to an algorithm with $\text{poly}(n, m) + O(n^2) \cdot 2^{O(\sqrt{n} \log n)} = \text{poly}(n, m) + 2^{O(\sqrt{n} \log n)}$ running time, where m is the total number of edges of the obstacles.

Our contribution. We are interested in the variant of TSP WITH OBSTACLES where the salesman is moving inside a simple polygon P with m edges in total, which contains the set S of points to be visited. In other words, there is a single obstacle which is the region outside the polygon. We call this variant TSP IN A SIMPLE POLYGON. From now on, we refer to the points in S as *sites*, to distinguish them from arbitrary points in P .

As mentioned, TSP WITH OBSTACLES in the plane can be solved in subexponential time using the algorithm of Marx, Pilipczuk, and Piliczuk for SUBSET TSP as a subroutine. Hence, TSP IN A SIMPLE POLYGON can be solved in $\text{poly}(n, m) + 2^{O(\sqrt{n} \log n)}$ time. Unfortunately, the algorithm of Marx, Pilipczuk, and Piliczuk for SUBSET TSP is complicated: the description of the algorithm and its correctness proof take 27 pages in total [23]. Here we only count the overview of the algorithm (Section 2, comprising 7 pages) and the detailed description of the algorithm and proof of correctness (Section 5, comprising 20 pages), and not the description of some of the tools being used (Sections 3 and 4, comprising 2.5 pages).

We present a much simpler algorithm with the same running time, based on the elegant algorithm by Hwang, Chang and Lee [17] for EUCLIDEAN TSP in the plane. We also prove several basic properties of optimal solutions for TSP WITH OBSTACLES, which are of independent interest.

2 Notation and basic properties

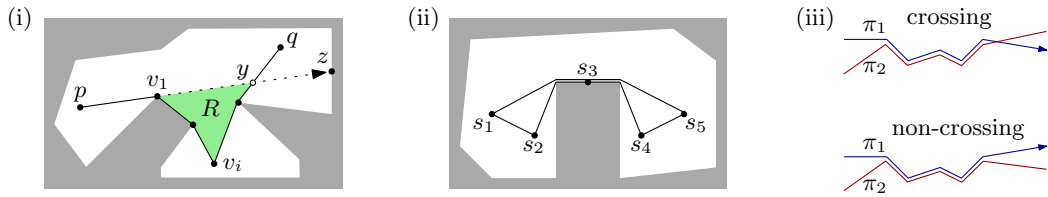
In this section we introduce the notation and terminology used throughout paper, and we prove several basic properties of optimal TSP tours in a simple polygon.

Let P be a simple polygon, which is the region in which the salesman can move. We consider P to be a closed set, so that shortest paths are well defined. Let $S = \{s_1, \dots, s_n\} \subset P$ denote the set of sites to be visited by the salesman. We say that a vertex v of P is *reflex* if the angle between the two edges incident to v , measured inside P , is more than 180 degrees.

Whenever we speak of a *path*, we mean a path that stays in P , unless stated otherwise. We denote the length of the line segment pq connecting points p and q by $|pq|$. Similarly, we denote the (Euclidean) length of a path π by $|\pi|$. For two points $p, q \in P$ we use $\pi(p, q)$ to denote the (unique) shortest path between them, that is, the path from p to q that has minimum length while staying inside P . Finally, for a path π and two points $a, b \in \pi$, the subpath of π between a and b is denoted by $\pi[a, b]$.

Consider a polygonal path π (that is, a path consisting of straight-line segments) and let v be a reflex vertex of P . We say that π *bends around* v if v coincides with an interior vertex (not an endpoint) of π and π is *locally shortest* at v . In other words, if e_1, e_2 are the two edges of π meeting at v , then the two edges of P meeting at v lie in the convex wedge defined by e_1 and e_2 . Observe that a shortest path π inside P must bend around a reflex vertex of P at each of the path's interior vertices. The next lemma shows that in a simple polygon, local optimality implies global optimality. It is folklore, but for completeness we give a proof.

5:4 TSP in a Simple Polygon



■ **Figure 1** (i) Illustration for the proof of Lemma 1. (ii) An optimal tour may pass through the same site twice, and can contain (partially) overlapping line segments. (iii) Crossing and non-crossing paths. Note: paths in parts (ii) and (iii) are shown slightly displaced where they overlap.

► **Lemma 1.** *Let p, q be two points inside a simple polygon P and let π be a path between p and q such that every interior vertex of π bends around a reflex vertex of P . Then, π is the shortest path from p to q in P .*

Proof. Let v_1, \dots, v_k be the interior vertices of π , so $\pi = (p, v_1, \dots, v_k, q)$. We will prove the lemma by induction on k . For $k = 0$ the lemma trivially holds, so assume $k > 0$.

Consider the ray from p through v_1 , and let z be the first point where the ray hits ∂P after v_1 . Then the segment v_1z splits P into two parts. Let P_1 be the part that contains p , and let P_2 be the other part. We claim that $q \in P_2$. Indeed, if this was not the case then π must cross v_1z at some point y . Consider the region $R \subset P$ enclosed by v_1y and $\pi[v_1, y]$; see Fig. 1(i). (Our assumptions do not immediately rule out that π intersects itself, so R need not be simple. But this does not invalidate the coming argument.) Since P is simple, R cannot contain any part of ∂P . Let v_i denote a vertex of π on $\pi[v_1, y]$ that has maximal distance from v_1y . Then the angle between the two edges of π incident to v_i has to be convex when measured inside R , otherwise v_i cannot be a vertex of maximal distance from v_1y . By assumption, π has to bend around a vertex of P at v_i , but this cannot happen since $R \subset P$. Hence, $q \in P_2$, as claimed.

We now observe that the shortest path from p to any point in P_2 passes through v_1 . Moreover, by the induction hypothesis we know that the path (v_1, \dots, v_k, q) is the shortest path from v_1 to q . We conclude that π is the shortest path from p to q in P . ◀

It is well known that optimal tours for EUCLIDEAN TSP in \mathbb{R}^2 are non-self-intersecting, that is, if $s_i s_j$ and $s_k s_t$ are non-consecutive edges in an optimal tour then $s_i s_j$ and $s_k s_t$ do not intersect (except when all sites are collinear). This is no longer true for TSP IN A SIMPLE POLYGON: two paths $\pi(s_i, s_j)$ and $\pi(s_k, s_t)$ that are part of an optimal tour can meet in one or more points, and sites may be visited more than once; see Fig. 1(ii). However, we can still formulate a non-crossing property for TSP WITH OBSTACLES, as shown next.

Let $\pi_1 = \pi(p_1, q_1)$ and $\pi_2 = \pi(p_2, q_2)$ be two shortest paths in P . Observe that their intersection $\pi_1 \cap \pi_2$, may contain at most one connected component, due to the uniqueness of shortest paths. Let γ denote this component. We give an arbitrary orientation to the path π_1 . We say that π_1 and π_2 are *crossing* when π_2 lies on opposite sides of π_1 just before γ and just after γ , relative to the chosen orientation of π_1 ; otherwise π_1 and π_2 are *non-crossing*. See Fig. 1(iii) for an example.

We now state the main result of this section.

► **Theorem 2.** *Let P be a simple polygon and let S be a set of n sites in P . Then there is an optimal tour T_{opt} through S such that*

1. T_{opt} passes at most twice through any point of P , and
2. any two paths of T_{opt} are non-crossing.



■ **Figure 2** Illustrations for the proof of Theorem 2.

► **Remark 3.** A site s_i may lie on the shortest path between two other sites s_j, s_k , in which case T_{opt} may use $\pi(s_j, s_k)$ plus two shortest paths with s_i as endpoint; see site s_3 in Fig. 1(ii). This does not contradict Property 1 of the theorem: T_{opt} still passes through s_i only twice.

Proof.

Proof of part 1. We first observe the following. Let r be a point through which T_{opt} passes at least twice. Give T_{opt} an arbitrary orientation, and suppose T_{opt} first arrives at r along a segment e_1 and later along a segment e_2 that is not collinear with e_1 . Then r must be a reflex vertex of P , and P must, locally at r , lie in the convex wedge defined by e_1 and e_2 ; otherwise we could shortcut T_{opt} at r as shown in Fig. 2(i).

Now suppose T_{opt} passes at least three times through some point r . Then there are at least three segments e_1, e_2, e_3 through which T_{opt} arrives at r . If two of these segments overlap then we can shortcut T_{opt} , which is a contradiction. Otherwise at least two of the three wedges defined by e_1, e_2, e_3 have an opening angle less than 180 degrees. The point r can be a reflex vertex of R , but locally at r , the polygon P can lie in only one of these two wedges; see Fig. 2(ii). Hence, we can shortcut at the other wedge, which is a contradiction.

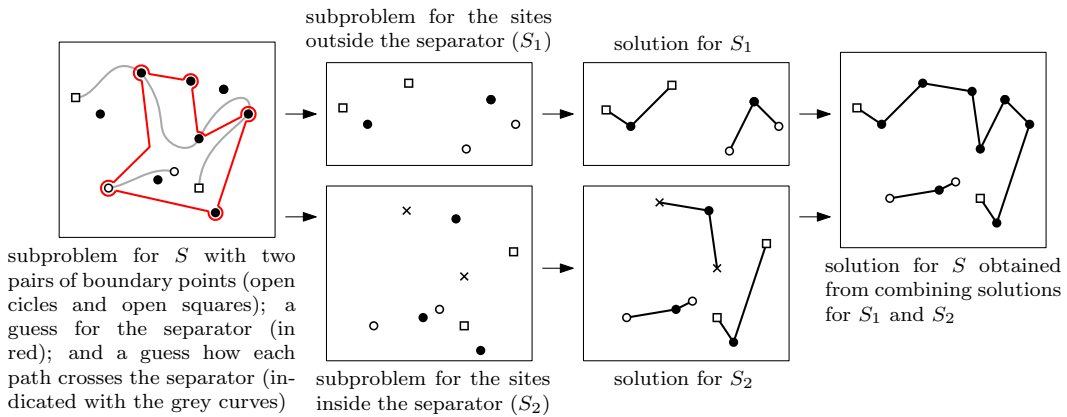
Proof of part 2. Number the sites in order along T_{opt} so that $T_{\text{opt}} = \bigcup_{\ell=1}^n \pi_\ell$, where we define $\pi_\ell := \pi(s_\ell, s_{\ell+1})$ and $s_{n+1} := s_1$. Suppose there are two paths π_i and π_j that cross each other. Notice that the paths $\pi(s_i, s_j)$ and $\pi(s_{i+1}, s_{j+1})$ do not cross each other; see Fig. 1, where the two crossing paths shown at the top can be “uncrossed”, resulting in the non-crossing paths shown at the bottom. Let Γ_1 be the part of T_{opt} from s_{i+1} to s_j and let Γ_2 be the part of T_{opt} from s_{j+1} to s_i . Then $T_{\text{opt}}^* := \Gamma_1 \cup \pi(s_{i+1}, s_{j+1}) \cup \Gamma_2 \cup \pi(s_i, s_j)$, where the direction of Γ_1 is reversed, is a tour of the same length as T_{opt} . Moreover, T_{opt}^* has fewer crossings; this is true because uncrossing π_i and π_j cannot generate new crossings by part 1 of the theorem. Hence, we can repeat the process until we are left with an optimal tour without any crossings. ◀

3 A subexponential algorithm for TSP IN A SIMPLE POLYGON

In this section we give a subexponential algorithm for TSP IN A SIMPLE POLYGON, based on the work by Hwang *et al.* [17]. The algorithm of Hwang *et al.* solves a problem that is slightly more general than TSP, so that it can be used in a recursive algorithm. In a recent paper, De Berg *et al.* [11] define the same generalized problem, but with a different terminology that will be more convenient for us. In what follows we will mostly use their terminology to define the problem, which De Berg *et al.* call EUCLIDEAN PATH COVER. (Hwang *et al.* called it GENERALISED EUCLIDEAN TSP).

Let $S' \subset S$ be a subset of n of the sites of the initial set of sites, let $B \subset S'$ be a set of *boundary sites*, and let M be a perfect matching on B . We say that a collection of paths² $\mathcal{P} = \{\pi_1, \pi_2, \dots, \pi_{|B|/2}\}$ *realizes* M on S' if (i) for each pair $(s_i, s_j) \in M$ there is a path

² Note that we are now temporarily back in the standard EUCLIDEAN TSP setting. Thus the paths mentioned here are not shortest paths between sites, but paths in the complete graph on the sites. In the current setting, the edges of these paths are straight-line segments between the corresponding sites; when we go back to TSP IN A SIMPLE POLYGON, they will be shortest paths between sites.



■ **Figure 3** An example of how the algorithm works. After trying all possible separators and all possible ways the paths cross the separator, we will have found the optimal solution.

$\pi_{ij} \in \mathcal{P}$ with s_i and s_j as endpoints, and (ii) the paths together visit each site in S' exactly once. The goal of EUCLIDEAN PATH COVER is to find a collection of paths of minimum total length that realizes M on S' . Note that we can solve EUCLIDEAN TSP by solving $n - 1$ instances of EUCLIDEAN PATH COVER on S' , namely for $B = \{s_1, s_i\}$ where $i \in \{2, \dots, n\}$, and taking the best solution found. Next we describe how an instance of EUCLIDEAN PATH COVER can be solved using the triangulation approach. For simplicity, and with a slight abuse of notation, we will denote the set of sites that need to be visited by S (rather than S').

The triangulation approach. Hwang *et al.* solve EUCLIDEAN PATH COVER with a separator-based divide-and-conquer algorithm, which we will refer to as the *triangulation approach*. It works as follows. We start by creating Q , a set of three points defining an arbitrarily large triangle containing all the points in S . Suppose we have a maximal triangulation \mathcal{T} of $S \cup Q$ that uses all segments from each path in an optimal solution \mathcal{P}_{opt} . Since \mathcal{T} is a maximal planar graph, Miller's separator theorem [25] implies that there exists a simple cycle \mathcal{C} in \mathcal{T} of at most $2\sqrt{2(n+3)}$ points from $S \cup Q$ such that at most $2(n+3)/3$ sites are inside \mathcal{C} , and at most $2(n+3)/3$ sites are outside \mathcal{C} . The idea is to use the separator \mathcal{C} to split the problem into two subproblems: one inside \mathcal{C} and one outside \mathcal{C} , and glue the solutions to these subproblems together to get the solution to the original problem. However, we have to make sure that the solutions inside and outside match with each other. To ensure this, we guess all possible ways in which \mathcal{P}_{opt} can cross \mathcal{C} , and handle each of them separately. Since the triangulation uses all segments from each path in \mathcal{P}_{opt} , no edge in \mathcal{P}_{opt} crosses an edge in \mathcal{C} . Hence, \mathcal{P}_{opt} only goes from inside \mathcal{C} to outside \mathcal{C} via nodes of \mathcal{C} . Thus guessing where \mathcal{P}_{opt} crosses \mathcal{C} amounts to guessing for each path in \mathcal{P}_{opt} at which nodes of \mathcal{C} it crosses \mathcal{C} (as well as the order of these crossing nodes). Each guess will lead to different pairs of subproblems to be solved inside and outside the separator. The optimal solution will then be the best solution over all guesses. See Fig. 3 for an example.

Above we assumed that we had a triangulation \mathcal{T} of $S \cup Q$ that uses all segments from each path in \mathcal{P}_{opt} . But we do not know \mathcal{P}_{opt} , since \mathcal{P}_{opt} is what we want to compute. We therefore try all possible simple cycles \mathcal{C} of at most $2\sqrt{2(n+3)}$ sites, and for each of them compute an optimal solution under the assumption that the solution does not use any edge that intersects an edge in the cycle. One of these guesses must correspond to the separator for

the (unknown) triangulation \mathcal{T} . Algorithm 1 gives a high-level description of the algorithm. Next we show how to apply the algorithm to TSP IN A SIMPLE POLYGON, and fill in the details for the various steps.

■ **Algorithm 1** *Triangulation-Approach*(S, B, M).

Input: set S of n sites; set $B \subseteq S$ of boundary points; perfect matching M on B

Output: A solution for EUCLIDEAN PATH COVER for the instance (S, B, M)

- 1: **if** $|S|$ is a sufficiently small constant **then**
 - 2: Compute an optimal solution \mathcal{P}_{opt} by brute-force
 - 3: **else**
 - 4: Generate all candidate separators \mathcal{C} for S , as explained in the text
 - 5: **for** each candidate separator \mathcal{C} **do**
 - 6: Generate all pairs of subproblems $(S_1, B_1, M_1), (S_2, B_2, M_2)$ for \mathcal{C} ,
as explained in the text. Recursively solve each pair of subproblems, and let
 $\mathcal{P}_{\text{opt}}(S_1, B_1, M_1, S_2, B_2, M_2)$ be the solution obtained by concatenating
the solutions to the subproblems.
 - 7: $\mathcal{P}_{\text{opt}} \leftarrow$ best of all solutions $\mathcal{P}_{\text{opt}}(S_1, B_1, M_1, S_2, B_2, M_2)$ from the previous step
 - 8: **return** \mathcal{P}_{opt}
-

3.1 Applying the triangulation approach to TSP IN A SIMPLE POLYGON

We now return to TSP IN A SIMPLE POLYGON. As above, we will solve a path-cover problem, which we call GENERALIZED EUCLIDEAN PATH COVER. This problem is the same as EUCLIDEAN PATH COVER, except that the edges used by the paths, which used to be straight-line segment connecting sites, are now going to be shortest paths connecting sites.

To apply the triangulation approach in our new setting, we need a maximal triangulation of an optimal solution \mathcal{P}_{opt} to the given instance of the GENERALIZED EUCLIDEAN PATH COVER. The existence of such a maximal triangulation implies that there is a small separator by Miller’s separator theorem, which we can then guess. There exists of course a triangulation of \mathcal{P}_{opt} if we are allowed to use all internal vertices of these paths as vertices in the triangulation: we just need to construct a so-called *constrained triangulation* on the set of all sites and internal shortest-path vertices, which uses the given segments on the paths. But the number of nodes of such a triangulation would not only depend on the number of sites, but also on the total complexity of the shortest paths, which depends on the complexity of the polygon P . The key idea to overcome this, is to work with a triangulation whose nodes are the sites in S and whose edges are shortest paths. Next, we show that such a triangulation always exists, and show how to create a maximal triangulation from this triangulation. After that we will describe the various steps of the algorithm in more detail, and we will prove its correctness and analyze its running time.

Triangulating \mathcal{P}_{opt} . Recall that a set $X \subseteq P$ is called *geodesically convex* if for any two points $p, q \in X$ the shortest path $\pi(p, q)$ in P is contained in X . The *relative convex hull* [32] of a set S of sites inside a simple polygon P , denoted by $\text{RCH}(S)$, is defined as the intersection of all geodesically convex sets containing S . Intuitively, $\text{RCH}(S)$ is obtained by placing a rubber band on ∂P , and then releasing the band so that it “snaps” around S , without crossing over any site and while staying inside P . The boundary of $\text{RCH}(S)$ consists of shortest paths connecting points of S ; see Fig. 4(i) for an illustration.

Let $E(S) := \{\pi(s_i, s_j) : s_i, s_j \in S \text{ and } i \neq j\}$ to be the set of all shortest paths between the sites in S . We now define an *sp-triangulation* (see Figure 4(ii)) of S to be a collection $\mathcal{T}_{\text{sp}}(S) \subset E(S)$ of pairwise non-crossing shortest paths between the sites in S such that

- $\mathcal{T}_{\text{sp}}(S)$ includes the shortest paths that form ∂RCH , and
- each face in the subdivision of $\text{RCH}(S)$ defined by $\mathcal{T}_{\text{sp}}(S)$ (after moving pieces of the paths slightly apart where they overlap) is bounded by exactly three paths.

► **Lemma 4.** *For any set S of sites inside a simple polygon P , and any given collection $E^* \subset E(S)$ of pairwise non-crossing paths, there exists an sp-triangulation $\mathcal{T}_{\text{sp}}(S)$ that includes the paths from E^* .*

Proof. Consider the subdivision of $\text{RCH}(S)$ induced by (the boundary of $\text{RCH}(S)$ and) the prescribed shortest paths in E^* . We will show how to triangulate each face F in this subdivision using paths from $E(S)$.

The face F has an outer boundary and it contains zero or more additional shortest paths. Let $E_{\text{out}}(F)$ denote the shortest paths forming the outer boundary, let $E_{\text{in}}(F)$ denote the remaining shortest paths, and let $E(F) := E_{\text{out}}(F) \cup E_{\text{in}}(F)$. Let $S_{\text{out}}(F)$ denote the sites from S on the outer boundary of F , let $S_{\text{in}}(F)$ denote the remaining sites, and let $S(F) = S_{\text{out}}(F) \cup S_{\text{in}}(F)$. If $S_{\text{in}}(F) = \emptyset$ and $|S_{\text{out}}(F)| = 3$ then F is already a triangle and we are done. Otherwise we will argue that there exist two sites $s_i, s_j \in S(F)$ such that

- the shortest path $\pi(s_i, s_j)$ lies inside F
- $\pi(s_i, s_j)$ is not yet present in $E(F)$ and
- $\pi(s_i, s_j)$ does not cross any shortest path in $E(F)$

These properties imply that we can add $\pi(s_i, s_j)$ to $E(F)$. This either splits F into two faces, or it adds a path to $E_{\text{in}}(F)$. In both cases we can continue recursively on the resulting face(s), until every face is a triangle. It remains to argue the existence of the pair s_i, s_j . We consider two cases:

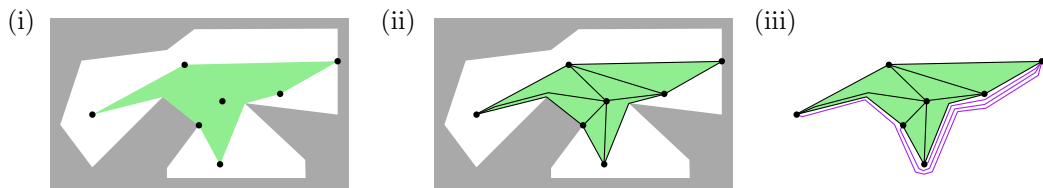
Case I: $S_{\text{in}}(F) = \emptyset$ and all sites from $S_{\text{out}}(F)$ are convex vertices of F .

Then take two sites $s_i, s_j \in S(F)$ that are not already connected by a shortest path in $E_{\text{out}}(F)$. Such a pair exists because $|S_{\text{out}}(F)| > 3$. Consider the shortest path π from s_i to s_j that is restricted to lie inside F . All interior vertices of π must bend around reflex vertices of F . Since all vertices from $S_{\text{out}}(F)$ are convex, these reflex vertices are not sites but reflex vertices of P . By Lemma 1, π must also be the shortest path in P between p and q and so $\pi \in E(S)$. Since π stays inside F , it does not cross any other shortest path.

Case II: $S_{\text{in}}(F) \neq \emptyset$ or $S_{\text{out}}(F)$ contains a site that is a reflex vertex of F .

We start by identifying a “reflex” site, from which we will then be able to add a path.

► **Claim.** There is a site $s_i \in S(F)$ with the following property: there is a line ℓ through s_i such that all paths from $E(F)$ that have s_i as an endpoint lie to the same side of ℓ , locally near s_i . (In other words, the edges of these paths that have s_i as an endpoint all lie to the same side of ℓ).



■ **Figure 4** (i) The relative convex hull of a set of points. (ii) An sp-triangulation. (iii) The corresponding maximal triangulation. Note: paths are shown slightly displaced where they overlap.

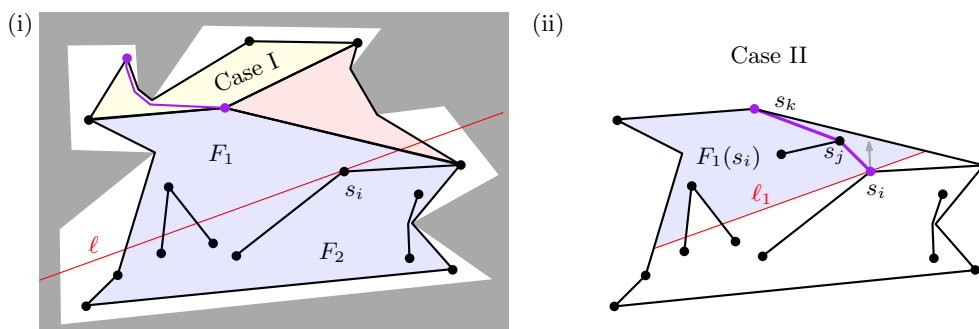


Figure 5 Illustrations for the proof of Lemma 4. (i) An example with three faces inside $\text{RCH}(S)$. The red face is already a triangle. The yellow face falls into Case 1 and can be triangulated by adding the purple path. The blue face falls into Case 2 of the proof. (ii) The ray from s_i hits a shortest path that has s_k as an endpoint in $F_1(s_i)$. The purple path is the shortest path from s_i to s_k inside $F_1(s_i)$, and therefore $\pi(s_i, s_j)$ is added to the triangulation.

Proof. First suppose that $S_{\text{in}}(F) = \emptyset$. Then $S_{\text{out}}(F)$ contains a site s_i that is a reflex vertex of F and, hence, has the required property. (Note that, except for the two paths on the outer boundary of F meeting at s_i , the site s_i has no other incident paths, since $S_{\text{in}}(F) = \emptyset$.)

Now suppose $S_{\text{in}}(F) \neq \emptyset$. Consider a connected component C in $E_{\text{in}}(F)$. (Or more precisely, a connected component of the set $\bigcup E_{\text{in}}(F) \subset \mathbb{R}^2$.) Then C may contain at most one site from $S_{\text{out}}(F)$ – indeed, if it would contain two such sites then it would split F and some paths in the component would be part of the outer boundary of F , a contradiction. Now see $E_{\text{out}}(F)$ as a simple polygon and consider the relative convex hull of $C \cap S(F)$ within this simple polygon. Then choose s_i to be any vertex of $\text{RCH}(C \cap (S(F)))$ that is a site from $S_{\text{in}}(F)$. Then there is a line ℓ through s_i such that the path edges incident to s_i lie to the same side of ℓ , namely a line ℓ that is (locally) tangent to $\text{RCH}(C \cap S(F))$. \triangleleft

Consider the site s_i and the line ℓ provided by the above claim. Let $\ell_1 \subset \ell$ denote the maximal subsegment of ℓ containing s_i and lying inside F . The segment ℓ_1 splits F in two pieces, F_1 and F_2 . Let F_2 denote the piece containing the edges incident to s_i and let F_1 denote the other piece. We claim that F_2 must contain all sites s_k already connected to s_i , that is, all sites $s_k \in S(F)$ such that $\pi(s_i, s_k) \in E(F)$. Indeed, if some neighbor s_k of s_i was in F_1 then $\pi(s_i, s_k)$ would intersect ℓ_1 at some point s and therefore we would be able to shorten $\pi(s_i, s_k)$ by using the straight segment $s_i s$ as a shortcut. Now, ℓ_1 together with the paths in $E_{\text{in}}(F)$ induces a subdivision of F into planar regions. Denote by $F_1(s_i)$ the region in this subdivision within F_1 that contains s_i on its boundary; see Fig. 5(ii), where this region is shown in purple.

Now shoot a ray from s_i into $F_1(s_i)$, and let π denote the first path from $E(F)$ hit by this ray. At least one of the two endpoints of π must be in $F_1(s_i)$, otherwise π would intersect ℓ_1 twice, contradicting that π is a shortest path. Let s_k be this endpoint. Consider the shortest path γ from s_i to s_k , restricted to lie in $F_1(s_i)$. Then γ has to bend around vertices of $F_1(s_i)$, which are either reflex vertices of P or sites in $S(F)$. Let s_j be the first vertex on γ that is a site; such a vertex exists, since the endpoint s_k of γ is a site. Then all interior vertices of $\gamma[s_i, s_j]$, the subpath of γ from s_i to s_j , bend around reflex vertices of P . By Lemma 1, the subpath $\gamma[s_i, s_j]$ is a shortest path in P . Hence, we have found a shortest $\pi(s_i, s_j)$ that we can add to $E(F)$. \blacktriangleleft

Note that $\mathcal{T}_{\text{sp}}(S)$, when viewed as a graph, may not be maximally planar. The reason is that the face outside $\text{RCH}(S)$ is not necessarily a triangle. However, we can easily turn $\mathcal{T}_{\text{sp}}(S)$ into a maximally planar graph, as follows. Let s_1, \dots, s_r be the sites on $\partial\text{RCH}(S)$, in the same order as they are encountered while following $\partial\text{RCH}(S)$ clockwise. For every pair (s_1, s_i) with $3 \leq i \leq r - 1$, we add the path from s_1 to s_i that follows ∂RCH clockwise, as an edge to our triangulation. We denote this set of “outside edges” by $\overline{E}(S)$; see the purple paths in Figure 4(iii) for an example. By adding these edges to $\mathcal{T}_{\text{sp}}(S)$, we triangulate the outer face and thus obtain a maximally planar graph, which we denote by $\mathcal{G}_{\text{sp}}(S)$.

Now that we have established the existence of a suitable triangulation, we can explain how to implement the various steps in Algorithm 1.

Preprocessing. As a preprocessing step we compute the shortest path $\pi(s_i, s_j)$ for each pair $s_i, s_j \in S$, which also gives us all pairwise distances. We then check for each pair of shortest paths whether they cross or not, so that later on in the algorithm we have this information readily available. Clearly, the preprocessing can be done in $\text{poly}(n, m)$ time. As it turns out, Algorithm 1 will not need the actual shortest paths or other geometric information – knowing pairwise distances (for solving the constant-size subproblems at the base of the recursion) and whether or not pairs of shortest paths cross (for larger subproblems) is all that is needed.

Generating all candidate separators. Miller’s theorem [25] guarantees that $\mathcal{G}_{\text{sp}}(S)$ has a simple-cycle separator of at most $2\sqrt{2n}$ nodes. The edges in this separator are edges in $\mathcal{G}_{\text{sp}}(S)$, which correspond to paths in $E(S) \cup \overline{E}(S)$. Note that for every pair of sites, there are at most two different paths in $E(S) \cup \overline{E}(S)$. We now generate our collection of candidate separators as follows:

- 1: **for** every circular sequence $s_1, s_2, \dots, s_t, s_1$ of at most $2\sqrt{2n}$ sites from S **do**
- 2: **for** every choice of paths from $E(S) \cup \overline{E}(S)$ to connect consecutive sites **do**
- 3: Check if the paths used in the resulting cycle \mathcal{C} are pairwise non-crossing and if the number of sites inside and outside \mathcal{C} is at most $2n/3$. If so, add \mathcal{C} to the collection of cycles.

Clearly, the collection of generated separators is guaranteed to contain the separator that would result from applying Miller’s theorem to $\mathcal{G}_{\text{sp}}(S)$. The total number of candidate separators is $n^{O(\sqrt{n})}$ (for choosing the circular sequence) times $2^{O(\sqrt{n})}$ (for choosing the paths to connect consecutive sites), so $n^{O(\sqrt{n})} \cdot 2^{O(\sqrt{n})} = 2^{O(\sqrt{n} \log n)}$ in total.

Note that for each candidate separator \mathcal{C} we need to check if it is simple, that is, if its paths are pairwise non-crossing. With the pre-computed information, this can be done in $O(|\mathcal{C}|^2) = O(n)$ time. Note that the paths from $\overline{E}(S)$ do not cross any other path by definition, so these paths need not be checked.

If \mathcal{C} is simple, we need to determine which sites are inside \mathcal{C} and which are outside \mathcal{C} . In fact, what is “inside” and what is “outside” is not important, we just need to partition the set of sites (that are not on the separator) into two subsets: the sites on one side of \mathcal{C} and the sites on the other side of \mathcal{C} . We can do that as follows.

Take any site s_i that is not on the separator \mathcal{C} , and consider another site s_j (that is not on \mathcal{C} either). We can decide if s_j is on the same side of \mathcal{C} as s_i by counting how often $\pi(s_i, s_j)$ crosses \mathcal{C} : site s_j is on the same side if and only if $\pi(s_i, s_j)$ crosses \mathcal{C} an even number of times. Because we determined in the preprocessing step for each pair of shortest paths whether they cross or not, we can do this counting in $O(\sqrt{n})$ time in total. The separator may also use paths from $\overline{E}(S)$ and these paths depend on the subproblem being solved. Hence, they have

not been considered in the preprocessing step. However, such paths cannot cross $\pi(s_i, s_j)$ by definition, so they can be ignored. This also means that we do not have to compute the paths in $\overline{E}(S)$ explicitly. We conclude that, for each candidate separator \mathcal{C} , we can partition the sites not on \mathcal{C} into subsets on either side of the separator in $O(n\sqrt{n})$ time.

Hence, in total we spend $O(n\sqrt{n}) \cdot 2^{O(\sqrt{n} \log n)} = 2^{O(\sqrt{n} \log n)}$ time to generate all possible separators and their corresponding partitions.

► **Lemma 5.** *The number of separators generated in Step 4 of Algorithm 1 is $2^{O(\sqrt{n} \log n)}$, and they can be generated in the same amount of time. This includes determining for each candidate \mathcal{C} a partitioning of $S \setminus \mathcal{C}$ into sites on one side of \mathcal{C} and on the other side of \mathcal{C} .*

Generating the subproblems for a given separator \mathcal{C} . This can be done as in the algorithms of Hwang *et al.* [17] and De Berg *et al.* [11]; see Fig. 3. For completeness we give a sketch.

Consider the set $\mathcal{C} \setminus B$, which contains the sites from \mathcal{C} that are not boundary sites. The path $\xi_{ij} \in \mathcal{P}_{\text{opt}}$ corresponding to a pair $(s_i, s_j) \in M$ may or may not visit sites of $\mathcal{C} \setminus B$. To generate our subproblems, we need to guess for every pair $(s_i, s_j) \in M$, which points of $\mathcal{C} \setminus B$ are visited by ξ_{ij} and in which order. (A site from $\mathcal{C} \setminus B$ should be used by at most one path ξ_{ij} .) Given such a guess, we can generate the corresponding subproblems, one for each side, as follows. Let $s_{k_1}, s_{k_2}, \dots, s_{k_t}$ be the ordered sequence of sites from $\mathcal{C} \setminus B$ that is our guess for where ξ_{ij} crosses \mathcal{C} . Define $\xi'_{ij} := s_i, s_{k_1}, s_{k_2}, \dots, s_{k_t}, s_j$. Then every pair of consecutive sites of ξ'_{ij} becomes a pair in the matching of one of the subproblems; which subproblem depends on which side we have guessed the corresponding part of ξ_{ij} to be on. After doing this for all ξ_{ij} corresponding to a pair in M , we have the matchings M_1 and M_2 of our subproblems. The sets of sites S_1 and S_2 for the recursive calls are generated as follows. First we add the sites inside \mathcal{C} to S_1 and the sites outside \mathcal{C} to S_2 . Then we add the sites of M_1 to S_1 , and the sites of M_2 to S_2 . Note that causes some sites to appear in both subproblems. Finally, the remaining sites on \mathcal{C} are all added to S_1 . In other words, we consider all points on \mathcal{C} to be on the same side of \mathcal{C} . See Fig. 3 for an example.

The total number of subproblems generated for a given separator \mathcal{C} is $n^{O(\sqrt{n})} = 2^{O(\sqrt{n} \log n)}$, since we have to guess for each separator node by which (if any) of the paths it is used and then choose an ordering for the crossings.

A proof of correctness. Next we prove that this gives an optimal solution.

► **Lemma 6.** *Let s_i be such that there is an optimal solution T_{opt} with non-crossing shortest paths that uses the path $\pi(s_1, s_i)$. Then $\text{Triangulation-approach}(S, B, M)$ with $B = \{s_1, s_i\}$ computes an optimal solution for TSP IN A SIMPLE POLYGON. Moreover, all other calls report valid solutions.*

Proof. We first argue that each reported solution is valid, by showing that any recursive call gives a valid solution of EUCLIDEAN PATH COVER. This is trivially true for the base case in the algorithm. The way in which the subproblems are generated and their solutions are combined, ensures that when the solutions to the subproblems are valid (which we can assume by induction) then the combined solution is valid. (Note: the generation of the subproblems and how they are combined is based on the original triangulation approach by Hwang *et al.* [17], so this part in fact follows from the correctness of their algorithm.)

Now consider a call $\text{Triangulation-approach}(S, B, M)$. Let $\mathcal{P}_{\text{opt}}(S, B, M)$ be an optimal solution to the subproblem. We will prove that if the parameters S, B, M are consistent with T_{opt} – that is, $\mathcal{P}_{\text{opt}}(S, B, M)$ is a subset of the global T_{opt} – then the algorithm computes an optimal solution to the subproblem. Since the initial call with $B = \{s_1, s_i\}$ is consistent

with T_{opt} by definition, this will prove the lemma. Note that if $\mathcal{P}_{\text{opt}}(S, B, M) \subset T_{\text{opt}}$, then $\mathcal{P}_{\text{opt}}(S, B, M)$ consists of non-crossing paths. Lemma 4 then implies that there exists an sp-triangulation $\mathcal{T}_{\text{sp}}(S)$ that includes the edges from $\mathcal{P}_{\text{opt}}(S, B, M)$. This can be extended to a maximally planar graph $\mathcal{G}_{\text{sp}}(S)$, which has separator of size at most $2\sqrt{2n}$. As argued earlier, this separator will be one of the generated candidate separators, and the way in which $\mathcal{P}_{\text{opt}}(S, B, M)$ crosses it will be corresponds to one of the generated subproblems. The parameters $S_{\text{in}}, B_{\text{in}}, M_{\text{in}}$ and $S_{\text{out}}, B_{\text{out}}, M_{\text{out}}$ of these subproblems are thus consistent with T_{opt} , and so we can assume by induction that they are solved optimally, thus leading to an optimal solution for the call to *Triangulation-approach*(S, B, M). ◀

Putting it all together. Lemma 6 gives the correctness of our algorithm so it remains to analyze the running time. The preprocessing takes $\text{poly}(n, m)$ time. The running time of *Triangulation-approach* satisfies the recurrence $T(n) = 2^{O(\sqrt{n} \log n)} + 2^{O(\sqrt{n} \log n)} \cdot T(\frac{2n}{3} + O(\sqrt{n}))$, which solves to $T(n) = 2^{O(\sqrt{n} \log n)}$. This leads to our final theorem.

► **Theorem 7.** TSP IN A SIMPLE POLYGON for a set S of n sites in a polygon P with m edges can be solved in $\text{poly}(n, m) + 2^{O(\sqrt{n} \log n)}$ time.

4 Conclusion

We introduced TSP IN A SIMPLE POLYGON, a natural variant of TSP that seems not to have been studied at all so far. The problem can be solved in $\text{poly}(n, m) + 2^{O(\sqrt{n} \log n)}$ time using a complicated algorithm of Marx, Pilipczuk, and Pilipczuk [24] as a subroutine. We presented a much simpler algorithm with the same running time. Our work raises several questions:

- It was recently shown that EUCLIDEAN TSP in \mathbb{R}^2 can be solved in $2^{O(\sqrt{n})}$ time [11]. Can we also get rid of the log-factor in the exponent for TSP IN A SIMPLE POLYGON?
- Can our approach be extended to polygons with holes? A major obstacle is that in this case a triangulation using shortest paths not always exists. We have been able to generalize our approach, by working with a suitable *collection* of paths between every pair of sites, but this significantly complicates matters, thus defeating the purpose.
- Can ideas from our approach be used to get a simplified solution to the more general SUBSET TSP problem for planar graphs?
- What about TSP WITH OBSTACLES in higher dimensions? Here neither our approach nor the approach by Marx, Pilipczuk, and Pilipczuk can be used.

References

- 1 Jeff Abrahamson and Ali Shokoufandeh. Euclidean TSP on two polygons. *Theor. Comput. Sci.*, 411(7-9):1104–1114, 2010. doi:10.1016/j.tcs.2009.12.003.
- 2 David L. Applegate, Robert E. Bixby, and Vasek Chvátal. *Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.
- 3 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.
- 4 Sanjeev Arora, Michelangelo Grigni, David R. Karger, Philip N. Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In Howard J. Karloff, editor, *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998)*, pages 33–41. ACM/SIAM, 1998. URL: <http://dl.acm.org/citation.cfm?id=314613.314632>.
- 5 Yair Bartal, Lee-Ad Gottlieb, and Robert Krauthgamer. The traveling salesman problem: Low-dimensionality implies a polynomial time approximation scheme. *SIAM J. Comput.*, 45(4):1563–1581, 2016. doi:10.1137/130913328.

- 6 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, 1962. doi:10.1145/321105.321111.
- 7 T.-H. Hubert Chan and Anupam Gupta. Approximating TSP on metrics with bounded global growth. *SIAM J. Comput.*, 41(3):587–617, 2012. doi:10.1137/090749396.
- 8 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.
- 9 William J. Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2011.
- 10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 11 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, and Sudeshna Kolay. An eth-tight exact algorithm for euclidean TSP. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 450–461, 2018. doi:10.1109/FOCS.2018.00050.
- 12 M. R. Garey, Ronald L. Graham, and David S. Johnson. Some NP-complete geometric problems. In *STOC*, pages 10–22. ACM, 1976.
- 13 Marcus Greiff and Anders Robertsson. Optimisation-based motion planning with obstacles and priorities. *IFAC-PapersOnLine*, 50(1):11670–11676, 2017. 20th IFAC World Congress. doi:10.1016/j.ifacol.2017.08.1677.
- 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 Gregory Gutin and Abraham P. Punnen. *The Traveling Salesman Problem and Its Variations*. Springer, 2002.
- 16 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM National Meeting*, ACM '61, pages 71.201–71.204, New York, NY, USA, 1961. ACM.
- 17 R. Z. Hwang, R. C. Chang, and Richard C. T. Lee. The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica*, 9(4):398–423, 1993. doi:10.1007/BF01228511.
- 18 Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.
- 19 Sándor Kisfaludi-Bak. A quasi-polynomial algorithm for well-spaced hyperbolic TSP. In *36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164 of *LIPICs*, pages 55:1–55:15, 2020. doi:10.4230/LIPICs.SoCG.2020.55.
- 20 Sándor Kisfaludi-Bak, Jesper Nederlof, and Karol Wegrzycki. A gap-eth-tight approximation scheme for euclidean TSP. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 351–362. IEEE, 2021. doi:10.1109/FOCS52979.2021.00043.
- 21 Philip N. Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM J. Comput.*, 37(6):1926–1952, 2008. doi:10.1137/060649562.
- 22 Philip N. Klein and Dániel Marx. A subexponential parameterized algorithm for subset TSP on planar graphs. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 1812–1830, 2014. doi:10.1137/1.9781611973402.131.
- 23 Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. On subexponential parameterized algorithms for steiner tree and directed subset TSP on planar graphs. *CoRR*, abs/1707.02190, 2017. arXiv:1707.02190.
- 24 Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. On subexponential parameterized algorithms for steiner tree and directed subset TSP on planar graphs. In *59th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2018)*, pages 474–484, 2018. doi:10.1109/FOCS.2018.00052.

- 25 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986. doi:10.1016/0022-0000(86)90030-9.
- 26 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999. doi:10.1137/S0097539796309764.
- 27 Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theor. Comput. Sci.*, 4(3):237–244, 1977.
- 28 Satish Rao and Warren D. Smith. Approximating geometrical graphs via “spanners” and “banyans”. In *STOC*, pages 540–550. ACM, 1998.
- 29 Gerhard Reinelt. *The Traveling Salesman, Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer, 1994. doi:10.1007/3-540-48661-5.
- 30 John Saalweachter and Zygmunt Pizlo. *Non-Euclidean Traveling Salesman Problem*, pages 339–358. Springer New York, New York, NY, 2008. doi:10.1007/978-0-387-77131-1_14.
- 31 Warren D. Smith and Nicholas C. Wormald. Geometric separator theorems & applications. In *FOCS*, pages 232–243. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743449.
- 32 Godfried Toussaint. Oan optimal algorithm for computing the relative convex hull of a set of points in a polygon. In *Proceedings of EURASIP, Signal Processing III: Theories and Applications (Part 2)*, pages 853–856, 1986.
- 33 Luca Trevisan. When hamming meets euclid: The approximability of geometric TSP and steiner tree. *SIAM J. Comput.*, 30(2):475–485, 2000. doi:10.1137/S0097539799352735.

Classical and Quantum Algorithms for Variants of Subset-Sum via Dynamic Programming

Jonathan Allcock ✉ 

Tencent Quantum Laboratory, Hong Kong, China

Yassine Hamoudi ✉ 

Simons Institute for the Theory of Computing, University of California, Berkeley, CA, USA

Antoine Joux ✉

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Felix Klingelhöfer ✉

G-SCOP, Université Grenoble Alpes, France

Miklos Santha ✉

Centre for Quantum Technologies and MajuLab, National University of Singapore, Singapore

Abstract

SUBSET-SUM is an NP-complete problem where one must decide if a multiset of n integers contains a subset whose elements sum to a target value m . The best known classical and quantum algorithms run in time $\tilde{O}(2^{n/2})$ and $\tilde{O}(2^{n/3})$, respectively, based on the well-known meet-in-the-middle technique. Here we introduce a novel classical dynamic-programming-based data structure with applications to SUBSET-SUM and a number of variants, including EQUAL-SUMS (where one seeks two disjoint subsets with the same sum), 2-SUBSET-SUM (a relaxed version of SUBSET-SUM where each item in the input set can be used twice in the summation), and SHIFTED-SUMS, a generalization of both of these variants, where one seeks two disjoint subsets whose sums differ by some specified value.

Given any modulus p , our data structure can be constructed in time $O(np)$, after which queries can be made in time $O(n)$ to the lists of subsets summing to any value modulo p . We use this data structure in combination with variable-time amplitude amplification and a new quantum pair finding algorithm, extending the quantum claw finding algorithm to the multiple solutions case, to give an $O(2^{0.504n})$ quantum algorithm for SHIFTED-SUMS. This provides a notable improvement on the best known $O(2^{0.773n})$ classical running time established by Mucha et al. [27]. We also study PIGEONHOLE EQUAL-SUMS, a variant of EQUAL-SUMS where the existence of a solution is guaranteed by the pigeonhole principle. For this problem we give faster classical and quantum algorithms with running time $\tilde{O}(2^{n/2})$ and $\tilde{O}(2^{2n/5})$, respectively.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Quantum algorithm, classical algorithm, dynamic programming, representation technique, subset-sum, equal-sum, shifted-sum

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.6

Related Version *Full Version*: <https://arxiv.org/abs/2111.07059> [3]

Funding This work has been supported by the European Union's H2020 Programme under grant agreement number ERC-669891, the ERA-NET Cofund in Quantum Technologies project QuantAlgo and the French ANR Blanc project RDAM. Research at CQT is funded by the National Research Foundation, the Prime Minister's Office, and the Ministry of Education, Singapore under the Research Centres of Excellence programme's research grant R-710-000-012-135.

Acknowledgements JA thanks Shengyu Zhang for helpful discussions during the course of this work.



© Jonathan Allcock, Yassine Hamoudi, Antoine Joux, Felix Klingelhöfer, and Miklos Santha; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, 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

1 Introduction

SUBSET-SUM is the problem of deciding whether a given multiset of n integers has a subset whose elements sum to a target integer m .

► **Problem 1** (SUBSET-SUM). Given a multiset $\{a_1, \dots, a_n\}$ of positive integers and a target integer m , find a subset $S \subseteq [n]$ such that $\sum_{i \in S} a_i = m$.

It is often useful to express SUBSET-SUM using inner product notation. We set $\bar{a} = (a_1, \dots, a_n) \in \mathbb{N}^n$, where the elements are taken in arbitrary order, and the task is to find $\bar{e} \in \{0, 1\}^n$ such that $\bar{a} \cdot \bar{e} = \sum_{i=1}^n a_i e_i = m$. The problem is famously NP-complete, and featured on Karp's list of 21 NP-complete problems [23] in 1972 (under the name of knapsack). It can be solved classically in time $\tilde{O}(2^{n/2})$ via the *meet-in-the-middle* technique [19]. Whether this problem can be solved in time $\tilde{O}(2^{(1/2-\delta)n})$, for some $\delta > 0$, is an important open question, but we know that the Exponential Time Hypothesis implies that SUBSET-SUM cannot be computed in time $m^{o(1)} 2^{o(n)}$ [13, 21]. SUBSET-SUM can also be solved in pseudopolynomial time, for instance in $O(nm)$ by a textbook dynamic programming approach, which was improved to a highly elegant $\tilde{O}(n+m)$ randomized algorithm by Bringmann [12]. However, assuming the Strong Exponential Time Hypothesis (SETH), it can be shown that for all $\epsilon > 0$, there exists $\delta > 0$, such that SUBSET-SUM cannot be computed in time $O(m^{1-\epsilon} 2^{n\delta})$ [2]. On a quantum computer, meet-in-the-middle can be combined with Quantum Search to solve SUBSET-SUM in time $\tilde{O}(2^{n/3})$.

1.1 Some Variants of Subset-Sum

SUBSET-SUM has several close relatives we will be concerned with in this paper.

► **Problem 2** (EQUAL-SUMS [32]). Given a set $\{a_1, \dots, a_n\}$ of positive integers, find two distinct subsets $S_1, S_2 \subseteq [n]$ such that $\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i$. In inner product notation we are looking for a nonzero vector $\bar{e} \in \{-1, 0, 1\}^n$ such that $\bar{a} \cdot \bar{e} = 0$.

The folklore classical algorithm [31] for EQUAL-SUMS runs in time $\tilde{O}(3^{n/2}) \leq O(2^{0.793n})$, and is also based on a meet-in-the-middle approach. In the classical case we arbitrarily partition the input into two sets of the same size, giving rise to vectors $\bar{a}_1, \bar{a}_2 \in \mathbb{N}^{n/2}$. Then we compute and sort the possible $3^{n/2}$ values $\bar{a}_1 \cdot \bar{e}$, for $\bar{e} \in \{-1, 0, 1\}^{n/2}$. Finally we compute the possible $3^{n/2}$ values of the form $\bar{a}_2 \cdot \bar{e}$ and, for each value, check via binary search if it has a collision (i.e. an item of the same value) in the first set of values. In the quantum case we use a different balancing, dividing the input into a set of size $n/3$ and a set of size $2n/3$, and then use Quantum Search over the larger set to find a collision. This folklore quantum algorithm has running time $\tilde{O}(3^{n/3}) \leq O(2^{0.529n})$. The classical running time of EQUAL-SUMS was reduced in a recent work by Mucha et al. [27] to $O(2^{0.773n})$, and it is an open problem whether this can be further improved.

A natural generalization of SUBSET-SUM is to allow each item in the input set to be used more than once in the summation, where the maximum number of times each item can be used is specified as part of the input to the problem. This is the analog of bounded knapsack, a well studied problem in the literature (see for example [24]). In particular, we will study the case when every item can be used at most twice.

► **Problem 3** (2-SUBSET-SUM). Given a multiset $\{a_1, \dots, a_n\}$ of positive integers and a target integer $0 < m < 2 \sum_{i=1}^n a_i$, find a vector $\bar{e} \in \{0, 1, 2\}^n$ such that $\bar{a} \cdot \bar{e} = m$.

There is a natural variant of SUBSET-SUM that generalizes both EQUAL-SUMS and 2-SUBSET-SUM. We call this variant SHIFTED-SUMS, whose investigation is the main subject of this paper.

► **Problem 4** (SHIFTED-SUMS). Given a multiset $\{a_1, \dots, a_n\}$ of positive integers and a shift $0 \leq s < \sum_{i=1}^n a_i$, find two distinct subsets $S_1, S_2 \subseteq [n]$ such that $\sum_{i \in S_1} a_i = s + \sum_{i \in S_2} a_i$.

The condition $S_1 \neq S_2$ is necessary in the case $s = 0$ to exclude the trivial solutions $S_1 = S_2$. The problem EQUAL-SUMS is a special case of SHIFTED-SUMS in this case, and it is easy to show (see the full version of the paper [3]) that 2-SUBSET-SUM can also be reduced to SHIFTED-SUMS without increasing the size of the input. This means that any algorithm for SHIFTED-SUMS automatically gives rise to an algorithm of the same complexity for EQUAL-SUMS and 2-SUBSET-SUM, and therefore we focus on constructing a quantum algorithm for SHIFTED-SUMS.

We additionally study the following variant of EQUAL-SUMS where, by the pigeonhole principle, a solution is guaranteed to exist. This search problem is total in the sense that its decision version is trivial because the answer is always “yes”. Such problems belong to the complexity class TFNP [26] consisting of NP-search problems with total relations. Problems in TFNP cannot be NP-hard unless NP equals co-NP. More precisely, the following problem belongs to the Polynomial Pigeonhole Principle complexity class PPP, defined by Papadimitriou [28], where the totality of the problem is syntactically guaranteed by the pigeonhole principle.

► **Problem 5** (PIGEONHOLE EQUAL-SUMS). Given a set $\{a_1, \dots, a_n\}$ of positive integers such that $\sum_{i=1}^n a_i < 2^n - 1$, find two distinct subsets $S_1, S_2 \subseteq [n]$ such that $\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i$.

There are 2^n subsets $S \subseteq [n]$. Since they all verify $0 \leq \sum_{i \in S} a_i \leq 2^n - 2$ there must exist two distinct subsets S_1, S_2 that sum to the same value, according to the pigeonhole principle.

1.2 Our Contributions and Techniques

We give new classical and quantum algorithms for SUBSET-SUM and several closely related problems. Our main contribution is a quantum algorithm for SHIFTED-SUMS (and for its special cases of EQUAL-SUMS and 2-SUBSET-SUM) that improves on the currently best known $O(2^{0.773n})$ classical algorithm [27] and on the folklore $O(2^{0.529n})$ quantum meet-in-the-middle algorithm. We also initiate the study of the PIGEONHOLE EQUAL-SUMS problem in the classical and quantum settings, where we obtain better complexities than what is known for the general EQUAL-SUMS problem.

► **Theorem** (Theorem 18 (Restated)). *There is a quantum algorithm for SHIFTED-SUMS that runs in time $O(2^{0.504n})$.*

► **Theorem** (Theorems 25, 26 (Restated)). *There are classical and quantum algorithms for PIGEONHOLE EQUAL-SUMS that run in time $\tilde{O}(2^{n/2})$ and $\tilde{O}(2^{2n/5})$, respectively.*

In the full version of the paper [3], we give new $\tilde{O}(2^{n/2})$ and $\tilde{O}(2^{n/3})$ classical and quantum algorithms for SUBSET-SUM, not based on the seminal meet-in-the-middle approach of Horowitz and Sahni [19]. We also describe partial results and potential directions for future work on modular versions of the problems studied in this paper.

At a high level, all of our algorithms use a *representation technique* approach. While this technique was originally designed to solve SUBSET-SUM when the instances are drawn from some specific distribution [20], here we follow the path of Mucha et al. [27] and use it in a

worst case analysis. Among our new algorithms, the quantization of this technique for the SHIFTED-SUMS problem is the most challenging and requires using several quantum tools. We will therefore explain first, via this algorithm, the difficulties we had to address and the methods we used to tackle them.

Shifted-Sums. The representation technique approach for SHIFTED-SUMS consists first of selecting a random prime $p \in \{2^{bn}, \dots, 2^{bn+1}\}$, where $b \in (0, 1)$ is some appropriate constant, and a random integer $k \in \{0, \dots, p-1\}$. Then we consider the *random bin* $T_{p,k}$, defined as $T_{p,k} = \{S \subseteq \{1, \dots, n\} : \sum_{i \in S} a_i \equiv k \pmod{p}\}$, and we search that bin and $T_{p,(k-s) \bmod p}$ for a colliding solution (i.e. $(S_1, S_2) \in T_{p,k} \times T_{p,(k-s) \bmod p}$ such that $\sum_{i \in S_1} a_i = s + \sum_{i \in S_2} a_i$). The choice of the bin size (which, on average, is roughly $2^{(1-b)n}$) should balance two opposing requirements: the bins should be sufficiently large to contain a solution and also sufficiently small to keep the cost of collision search low.

To satisfy the above two requirements, our algorithm uses the concept of a *maximum solution*. This is the maximum of $|S_1| + |S_2|$, when S_1, S_2 are disjoint and form a solution. Let this maximum solution size be ℓn , for some $\ell \in (0, 1)$. The algorithm consists of two different procedures, designed to handle different maximum solution sizes. For ℓ close to 0 or close to 1, the quantization of the meet-in-the-middle method adapted to solutions of size ℓn is used because it performs better. In this case, the quantization does not present any particular difficulties: it is a straightforward application of Quantum Search with the appropriate balancing. We therefore focus the discussion on the representation technique procedure used for values of ℓ away from 0 or 1. When S_1, S_2 form a maximum solution of size ℓn then, for every $X \subseteq \overline{S_1 \cup S_2}$, the pairs $S_1 \cup X, S_2 \cup X$ also form a solution, and all these solutions have different values (see Lemma 20). This makes it possible to bound from below, not only the number of solutions, but also the number of *solution values* by $2^{(1-\ell)n}$, which makes the use of the representation technique successful.

The most immediate way to quantize the procedure is to replace classical collision finding by the quantum element distinctness algorithm of Ambainis [4]. However, in a straightforward application of this algorithm we face a difficulty. For concreteness, we explain this when $\ell = 3/5$. In that case, by the above, the total number of solutions with different values is at least $2^{2n/5}$. This is handy for applying quantum element distinctness: we can select a random prime $p \in \{2^{2n/5}, \dots, 2^{2n/5+1}\}$ and expect to have a solution in the random bin $T_{p,k}$ with reasonable probability. The expected size $|T_{p,k}|$ of the bin is about $2^{3n/5}$, and therefore the running time of Ambainis' algorithm should be of the order of $|T_{p,k}|^{2/3}$ which is also about $2^{2n/5}$. However, the quantum element distinctness algorithm requires us to perform queries to $T_{p,k}$. That is, for some indexing $T_{p,k} = \{S_1, \dots, S_{|T_{p,k}|}\}$ of the elements of $T_{p,k}$, we need to implement the oracle $O_{T_{p,k}}|I\rangle|0\rangle = |I\rangle|S_I\rangle$, where $1 \leq I \leq |T_{p,k}|$. In other words, given $1 \leq I \leq |T_{p,k}|$, we have to be able to find the I th element in $T_{p,k}$ (for some ordering of that set). In the usual description of the element distinctness algorithm there is a simple way to do that (for example, the set over which the algorithm is run is just a set of consecutive integers). However, finding a simple bijection among the first $|T_{p,k}|$ integers and $T_{p,k}$ is not a trivial task. Unlike in the classical case, explicitly enumerating $T_{p,k}$ is not an option because this would take too long, requiring about $2^{3n/5}$ time steps. Instead, we use dynamic programming to compute the table $t_p[i, j] = |\{S \subseteq \{1, \dots, i\} : \sum_{s \in S} a_s \equiv j \pmod{p}\}|$. Computing the cardinality of the bins is cheaper than computing their contents, and can be done in time $O(np) = \tilde{O}(2^{2n/5})$. Crucially, once the table is constructed, one can deduce the paths through it that led to $t_p[n, k] = |T_{p,k}|$, in order to find each element of $T_{p,k}$ in time $O(n)$. More precisely, we define a strict total order \prec over $\mathcal{P}([n])$ and prove:

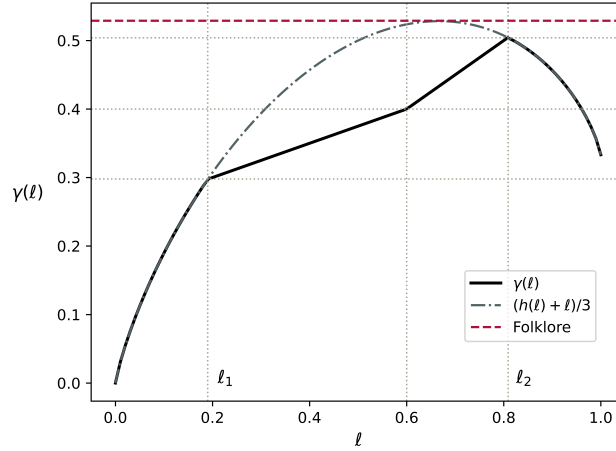
► **Theorem** (Theorem 11 (Restated)). *Let $T_{p,k}$ be enumerated as $T_{p,k} = \{S_1, \dots, S_{|T_{p,k}|}\}$ where $S_1 \prec \dots \prec S_{|T_{p,k}|}$. Given any integer $I \in \{1, \dots, |T_{p,k}|\}$ and random access to the elements of the table t_p , the set S_I can be computed in time $O(n)$.*

This novel data structure will be used in our algorithms for SUBSET-SUM (see the full version [3]), SHIFTED-SUMS and PIGEONHOLE EQUAL-SUMS. We now describe the additional quantum tools we use for SHIFTED-SUMS. The algorithm randomly chooses a bin size of about $2^{(1-b)n}$ where b is defined differently depending on whether ℓ is above or below $3/5$, as different tools are required in these two regions. When $\ell \leq 3/5$, with high probability a random bin contains multiple solutions from which we can profit. To that end, we construct a quantum algorithm for finding a pair marked by a binary relation $\mathcal{R}(x, y) := \mathbb{1}_{f(x)=g(y)}$ that tests if two values $f(x)$ and $g(y)$ are equal. Our algorithm generalizes the element distinctness [4] and claw finding [30] algorithms to the case of multiple marked pairs. Using an appropriate variant of the birthday paradox (see Lemma 5) we prove:

► **Theorem** (Theorem 6 (QUANTUM PAIR FINDING - Restated)). *Consider two sets of $N \leq M$ elements, respectively, and an evaluation function on each set. Suppose that there are K disjoint pairs in the product of the two sets such that in each pair the elements evaluate to the same value. There is a quantum algorithm that finds such a pair in time $\tilde{O}((NM/K)^{1/3})$ if $N \leq M \leq KN^2$ and $\tilde{O}((M/K)^{1/2})$ if $M \geq KN^2$.*

The best complexity when $\ell \leq 3/5$ is then obtained by choosing the bin size parameter b as a function of ℓ , which balances the cost of the construction of the dynamic programming table and the Quantum Pair Finding. When $\ell > 3/5$, choosing a bin size $2^{(1-b)n}$, for $b \leq 1 - \ell$, guarantees that a random bin contains at least one solution with high probability. However, a better running time at first seems to be achievable by the following argument: Choose $b > 1 - \ell$, for which there is exponentially small probability that a random bin contains a solution, and use amplitude amplification to boost the success probability. Balancing again the dynamic programming and Quantum Pair Finding costs would then give an optimal bin size of $2^{3n/5}$, independent of ℓ . However, this argument contains a subtlety. Standard amplitude amplification requires that the random bins $T_{p,k}$ simultaneously satisfy two conditions: beside containing a solution, they should also have sizes close to the expected size of about $2^{(1-b)n}$. But there is no guarantee that these two events coincide, and a priori it could be that the exponentially small fraction of $T_{p,k}$ containing a solution also happen to have sizes that far exceed the expectation. Fortunately, by carefully bounding the expectation of the product of bin sizes, we can use the Variable-Time Amplitude Amplification algorithm of Ambainis [5], and achieve the same running time as given by the above argument. We believe that this a nice and natural application of this method. The running time of our algorithm for SHIFTED-SUMS, as a function of ℓ , is shown in Fig. 1.

Pigeonhole Equal-Sums. This problem can be solved by any (classical or quantum) algorithm which solves the general EQUAL-SUMS (or SHIFTED-SUMS) problem. However, one can make use of the explicit promise of $a_1 + \dots + a_n < 2^n - 1$ to design faster algorithms than provided for by the general case when $\ell > 3/5$. Indeed, by the pigeonhole principle, for any value of p , if a bin $T_{p,k}$ has size larger than $2^n/p$ then it must contain a solution (see Lemma 24). Moreover, there must exist at least one such oversized bin. The array t_p can now be constructed both for *locating the index k* of one oversized bin and searching for a solution in it. We thus obtain a classical algorithm running in time $\tilde{O}(p + 2^n/p)$, and a quantum algorithm running in time $\tilde{O}(p + (2^n/p)^{2/3})$. These two quantities are minimized by *deterministically* choosing $p = 2^{n/2}$ and $p = 2^{2n/5}$ respectively (see Section 5).



■ **Figure 1** Running time exponent $\gamma(\ell)$ of the quantum SHIFTED-SUMS algorithm, as a function of the maximum solution ratio ℓ (see Theorem 18). The maximum value of $\gamma(\ell)$ is ≈ 0.504 which occurs at $\ell = \ell_2 \approx 0.809$. For reference, the curve $(h(\ell) + \ell)/3$ corresponding to Theorem 23 is plotted for all values of ℓ , as is the value of 0.529 corresponding to the exponent of the folklore (quantized) meet-in-the-middle algorithm, as applied to SHIFTED-SUMS.

1.3 Related Works

The closest work to our contribution is the paper of Mucha et al. [27] solving EQUAL-SUMS classically in time $O(2^{0.773n})$. Their algorithm and ours use the same two basic procedures, based respectively on the meet-in-the-middle method and the representation technique. Let us point out some of the differences. Unlike our algorithm which is based around the concept of the size of a maximum solution, the classical algorithm is analyzed as function of a *minimum size* solution, defined as $|S_1| + |S_2|$, where this sum is minimized over all solutions. The use the classical algorithm makes from a minimum solution S_1, S_2 of size ℓn is that when $\ell > 1/2$ the number of *solution values* can be bounded from below by $2^{(1-\ell)n}$. However, this does not hold for 2-SUBSET-SUM when ℓn is the size of a minimum solution, but *is* valid for both EQUAL-SUMS and 2-SUBSET-SUM when it is the size of a maximum solution. Another difference with [27] is that their classical representation technique algorithm always samples p from the same set $\{2^{(1-\ell)n}, \dots, 2^{(1-\ell)n+1}\}$, while we randomly choose $p \in \{2^{bn}, \dots, 2^{bn+1}\}$ where b is defined differently depending on in which of two distinct regions ℓ lies. This makes it possible to use different quantum techniques in these two regions.

The representation technique was designed by Howgrave-Graham and Joux [20] to solve random SUBSET-SUM instances under some hypotheses (heuristics) about how such instances behave during the run of the algorithm. The idea is to decompose a single solution to the initial problem into many distinct decompositions of a sum of half-solutions. To compensate for this blow-up, an additional linear constraint is added to select approximately one of these decompositions. Under some rather strong assumptions, which are satisfied for a large fraction of randomly chosen instances, [20] can solve SUBSET-SUM instances in time $O(2^{0.337n})$. Since then, several variants of this classical method have been proposed [8, 15, 10, 14], while others have investigated quantum algorithms based on the representation technique. Bernstein et al. [9] improved on [20] using quantum walks, and their algorithm (again under some hypotheses) runs in time $O(2^{0.242n})$. Further quantum improvements were made in this

context by [18] and [10]. However, we emphasize that the algorithms in all these papers work for random inputs generated from some distributions. The paper [27] gave the first classical algorithm based on the representation technique that works for worst case inputs and with proven bounds. To our knowledge, for worst case inputs with provable guarantees, the first quantum algorithm based on the representation technique is given in our work.

Dynamic programming is notoriously hard to quantize, with a key obstacle being the intrinsically sequential way in which the solution to a large problem is constructed from the solutions to smaller subproblems. Certain basic dynamic programming algorithms can be trivially accelerated by Quantum Search or Minimum Finding (see, e.g. [1]) but beyond that few other quantum improvements are known. One notable exception is the work of Ambainis et al. [6] who gave faster quantum algorithms for several NP-hard problems for which the best classical algorithms use dynamic programming. Their algorithms precompute solutions for smaller instances via dynamic programming and then use non-trivial Quantum Search recursively on the rest of the problem. In our work, the dynamic programming subroutine that we use is classical (although for SHIFTED-SUMS it is performed in superposition) and the sequential nature of the process is therefore not an issue. Rather than using quantum computing to accelerate classical dynamic programming, we instead use dynamic programming to enable fast queries, required for Quantum Search and Pair Finding, to be performed on complicated sets.

The class PPP is arguably less studied than other syntactically definable subclasses of TFNP, such as PLS (Polynomial Local Search) and PPA (Polynomial Parity Argument), and it is not known whether PIGEONHOLE EQUAL-SUMS is complete in PPP. In fact, the first complete problem for the class was only identified relatively recently [29]. Our results for PIGEONHOLE EQUAL-SUMS suggest that the problem is indeed simpler to solve than EQUAL-SUMS. In spirit, a similar result was obtained in [7] where it was shown that, for an optimization problem closely related to EQUAL-SUMS, better approximation schemes can be obtained for instances with guaranteed solutions.

1.4 Structure of the Paper

In Section 2.1, we define the notations and provide some basic facts. The quantum computational model and the algorithmic primitives used in the paper, such as our Quantum Pair Finding algorithm, are given in Section 2.2. In Section 3.1, we introduce the dynamic programming data structure and show how it can be used to implement fast subset-sum queries. The properties of this data structure needed for our work are proved in Section 3.2. The main application to the SHIFTED-SUMS problem is described in Section 4. Finally, we study the pigeonhole variant of EQUAL-SUMS in Section 5.

2 Preliminaries

2.1 Notations and Basic Facts

We use the $\tilde{O}(x)$ and $\tilde{\Omega}(x)$ notations to hide factors that are polylogarithmic in the argument x . For integers $0 \leq m < n$, we denote by $[m..n]$ the set $\{m, m+1, \dots, n\}$, and by $[n]$ the set $[1..n]$. For sets $S, S' \subseteq [n]$ we denote by \bar{S} the set $[n] \setminus S$, and by $S \Delta S'$ the symmetric difference of S and S' . Given a multiset $A = \{a_1, \dots, a_n\}$ and subset $S \subseteq [n]$ we denote $\Sigma_A(S) := \sum_{i \in S} a_i$. When the set A is clear from the context, we will omit the subscript and simply denote the subset sum by $\Sigma(S)$. The power set of $[n]$ will be denoted by $\mathcal{P}([n]) := \{S : S \subseteq [n]\}$. For arbitrary integers a and b and a modulus p we say that a is

congruent to b modulo p , and we write $a \equiv b \pmod{p}$ or $a \equiv_p b$ if $a - b$ is divisible by p . By $a \pmod{p}$ we denote the unique integer in $\{0, \dots, p - 1\}$ which is congruent to a modulo p . The binary entropy function will be denoted by $h(x) = -x \log_2(x) - (1 - x) \log_2(1 - x)$.

► **Fact 1** ([16], page 530). *For every constant $\ell \in (0, 1)$ and for every large enough integer n , the following bounds hold: $\frac{2^{nh(\ell)}}{\sqrt{8n\ell(1-\ell)}} \leq \binom{n}{\ell n} < \frac{2^{nh(\ell)}}{\sqrt{2\pi n\ell(1-\ell)}}$.*

► **Fact 2.** *Let $b > 0$ be a constant, n a large enough integer and $a_1 \neq a_2$ two integers. Then, for a random prime $p \in [2^{bn} \dots 2^{bn+1}]$ we have $\Pr_p[a_1 \equiv_p a_2] \leq \frac{\log(|a_1 - a_2|)}{2^{bn}}$.*

Proof. The number of primes that belong to the interval $[2^{bn} \dots 2^{bn+1}]$ is at least $2^{bn}/bn$ for n large enough (see [17, p.371]). Moreover, there are at most $\frac{\log(|a_1 - a_2|)}{bn}$ prime numbers larger than 2^{bn} that divide $a_1 - a_2$. The result follows by a union bound. ◀

2.2 Quantum Algorithms

Quantum Computational Model. Similar to previous works on quantum element distinctness [4], quantum dynamic programming [6] and quantum subset sum algorithms [9], in our quantum algorithm running time analysis we assume the standard circuit model (where computational time corresponds to the number of single and two qubit gates) augmented with random access to quantum memory. That is, coherent access to any element of an m -qubit array can be performed in time polylogarithmic in m . Note that fully quantum memory is required in Algorithm 2 for SHIFTED-SUMS since multiple bins $T_{p,k}$ must be computed and stored in superposition.

Algorithmic primitives. We use the following generalization of Grover's search to the case of an unknown number of solutions.

► **Fact 3** (QUANTUM SEARCH, Theorem 3 in [11]). *Consider a function $f : [N] \rightarrow \{0, 1\}$ with an unknown number $K = |f^{-1}(1)|$ of marked items. Suppose that f can be evaluated in time τ . Then, the Quantum Search algorithm finds a marked item in f in expected time $\tilde{O}(\sqrt{N/K} \cdot \tau)$.*

Given a classical subroutine with stopping time τ that returns a marked item with probability ρ , we can convert it into a constant success probability algorithm with expected running time $O(\mathbb{E}[\tau]/\rho)$ by repeating it $O(1/\rho)$ times. Ambainis proved a similar result for the case of quantum subroutines, with a dependence on the second moment of the stopping time τ , and a Grover-like speed-up for the dependence on ρ .

► **Fact 4** (VARIABLE-TIME AMPLITUDE AMPLIFICATION, Theorem 2 in [5]). *Let \mathcal{A} be a quantum algorithm which looks for a marked element in some set. Let τ be the random variable corresponding to the stopping time of the algorithm, and let ρ be its success probability. Then the Variable-Time Amplitude Amplification algorithm finds a marked element in the above set with constant success probability in maximum time $\tilde{O}(\sqrt{\mathbb{E}[\tau^2]/\rho})$.*

The next result is a variant of the Birthday paradox over a product space $[N] \times [M]$, where at least K disjoint pairs are marked by some binary relation \mathcal{R} . Two pairs (x, y) and (x', y') are said to be disjoint if $x \neq x'$ and $y \neq y'$. The disjointness assumption is made to simplify the analysis and will be satisfied in our applications.

► **Lemma 5** (VARIANT OF THE BIRTHDAY PARADOX). *Consider three integers $1 \leq K \leq N \leq M$. Let $\mathcal{R} : [N] \times [M] \rightarrow \{0, 1\}$ be a binary relation such that there exist at least K mutually disjoint pairs $(x_1, y_1), \dots, (x_K, y_K) \in [N] \times [M]$ with $\mathcal{R}(x_k, y_k) = 1$ for all $k \in [K]$. Given an integer $r \leq O(\sqrt{NM/K})$, define $\epsilon(r)$ to be the probability of obtaining both elements from at least one marked pair when r numbers from $[N]$ and r numbers from $[M]$ are chosen independently and uniformly at random. Then, $\epsilon(r) \geq \Omega\left(\frac{r^2 K}{NM}\right)$.*

Proof. Fix any K disjoint marked pairs $(x_1, y_1), \dots, (x_K, y_K)$. Let X_1, \dots, X_r (resp. Y_1, \dots, Y_r) be r independent and uniformly distributed random variables over $[N]$ (resp. $[M]$). For any indices i, j , let $Z_{i,j}$ denote the binary random variable that takes value 1 if $\{X_i, X_j\}$ is one of the K fixed pairs, and set $Z = \sum_{i,j} Z_{i,j}$. By definition, we have $\epsilon(r) \geq \Pr[Z \neq 0]$. We lower bound this quantity by using the inclusion-exclusion principle,

$$\epsilon(r) \geq \sum_{i,j} \Pr(Z_{i,j} = 1) - \frac{1}{2} \sum_{(i,j) \neq (k,\ell)} \Pr(Z_{i,j} = 1 \wedge Z_{k,\ell} = 1).$$

The first term on the right-hand side is equal to $\sum_{i,j} \Pr(Z_{i,j} = 1) = \frac{r^2 K}{NM}$. For the second term, the analysis depends on whether the indices i, k and j, ℓ are distinct or not. If they are distinct then $\Pr(Z_{i,j} = 1 \wedge Z_{k,\ell} = 1) = \frac{K^2}{(NM)^2}$ since $Z_{i,j}$ and $Z_{k,\ell}$ are independent. Otherwise, suppose for instance that $i = k$. Since the K fixed pairs are disjoint, we have $\Pr(Z_{i,j} = 1 \wedge Z_{i,\ell} = 1) = \Pr(Z_{i,j} = 1 \wedge Y_j = Y_\ell) = \frac{K}{NM^2}$. Finally, there are $4\binom{r}{2}^2$ ways of choosing the indices i, j, k, ℓ when $i \neq k$ and $j \neq \ell$, and $4r\binom{r}{2}$ ways when $i = k$ or $j = \ell$. By putting everything together we obtain that, $\epsilon(r) \geq \frac{r^2 K}{NM} - 2\binom{r}{2}^2 \frac{K^2}{(NM)^2} - 2r\binom{r}{2} \frac{K}{NM^2} \geq \Omega\left(\frac{r^2 K}{NM}\right)$. ◀

We use the above result to construct a quantum algorithm for finding a marked pair when the relation $\mathcal{R}(x, y)$ is determined by checking if two underlying values $f(x)$ and $g(y)$ are equal or not. Our analysis essentially generalizes the quantum element distinctness [4] and claw finding [30] algorithms to the case of $K > 1$.

► **Theorem 6** (QUANTUM PAIR FINDING). *There is a bounded-error quantum algorithm with the following properties. Consider four integers $1 \leq K \leq N \leq M \leq R$ with $R \leq N^{O(1)}$. Let $f : [N] \rightarrow [R]$ and $g : [M] \rightarrow [R]$ be two functions that can be evaluated in time τ . Define $\mathcal{R} : [N] \times [M] \rightarrow \{0, 1\}$ to be any of the two following binary relations:*

1. $\mathcal{R}(x, y) = 1$ if and only if $f(x) = g(y)$.
2. $\mathcal{R}(x, y) = 1$ if and only if $f(x) = g(y)$ and $x \neq y$.

Suppose that there exist at least K mutually disjoint pairs $(x, y) \in [N] \times [M]$ such that $\mathcal{R}(x, y) = 1$. Then, the algorithm returns one such pair in time $\tilde{O}((NM/K)^{1/3} \cdot \tau)$ if $N \leq M \leq KN^2$ and $\tilde{O}((M/K)^{1/2} \cdot \tau)$ if $M \geq KN^2$.

Proof. If $N \leq M \leq KN^2$ the algorithm consists of running a quantum walk over the product Johnson graph $J(N, r) \times J(M, r)$ with $r = (NM/K)^{1/3}$. This walk has spectral gap $\delta = \Omega(1/r)$ and the fraction ϵ of vertices containing both elements from at least one marked pair satisfies $\epsilon \geq \Omega\left(\frac{r^2 K}{NM}\right)$ by Lemma 5. Using the MNRS framework [25], the query complexity of finding one marked pair is then $O(S + \frac{1}{\sqrt{\epsilon}}(\frac{U}{\sqrt{\delta}} + C))$, where the setup cost is $S = r$, the update cost is $U = O(1)$, and the checking cost is $C = 0$. This leads to a query complexity of $O(r + 1/\sqrt{\epsilon\delta}) = O((NM/K)^{1/3})$. By a simple adaptation of the data structures described in [4, Section 6.2] or [22, Section 3.3.4], this can be converted to a similar upper bound on the time complexity with a multiplicative overhead of τ .

If $M \geq KN^2$, the algorithm instead stores all pairs $\{(x, f(x))\}_{x \in [N]}$ in a table – sorted according to the value of the first coordinate – and then runs the Quantum Search algorithm on the function $F : [M] \rightarrow \{0, 1\}$ where $F(x') = 1$ if there exists $x \in [N]$ such that $\mathcal{R}(x, y) = 1$.

There are at least K marked items and F can be evaluated in time $O(\tau + \log N)$ using the sorted table. Thus, the running time is $\tilde{O}(N \cdot \tau + (M/K)^{1/2} \cdot (\tau + \log N)) = \tilde{O}((M/K)^{1/2})$ by Fact 3. ◀

3 Dynamic Programming Data Structure

3.1 Construction of the Data Structure

Here we introduce our dynamic programming data structure, and show how it can be used to implement low cost subset-sum queries to the elements of the set $T_{p,k}$ defined as follows.

► **Definition 7.** Let $A = \{a_1, \dots, a_n\}$ be a multiset of n integers. For integers $p \geq 2$ and $k \in \{0, 1, \dots, p-1\}$, define the set $T_{p,k}$ by $T_{p,k} = \{S \subseteq [n] : \Sigma_A(S) \equiv k \pmod{p}\}$, and denote the cardinality of $T_{p,k}$ by $t_{p,k} := |T_{p,k}|$.

Our main tool is the table t_p , defined below, constructed by dynamic programming. As $t_{p,k} = t_p[n, k]$, once the table is constructed, the size $t_{p,k}$ of $T_{p,k}$ can be read off the last row.

► **Lemma 8.** Let n, p be non-negative integers. In time $O(np)$, the $(n+1) \times p$ table $t_p[i, j] = |\{S \subseteq \{1, \dots, i\} : \Sigma(S) \equiv j \pmod{p}\}|$ can be constructed by dynamic programming, where $i \in [0..n]$ and $j \in [0..p-1]$.

Proof. To compute the elements of the table, observe that $t_p[0, 0] = 1$ and $t_p[0, j] = 0$ for $j > 0$. The remaining elements can be deduced from the relation $t_p[i, j] = t_p[i-1, j] + t_p[i-1, (j-a_i) \bmod p]$. The i^{th} row of t_p can thus be deduced from the $(i-1)^{\text{th}}$ row and a_i in time $O(p)$ and the computation of all rows can be completed in time $O(np)$. ◀

We now show how to use the table t_p to quickly query any element of $T_{p,k}$. To do so, we first define an ordering of the elements of $T_{p,k}$.

► **Definition 9.** Let \prec be the relation over $\mathcal{P}([n])$ defined as follows: for all $S_1, S_2 \subseteq [n]$, $S_1 \prec S_2$ if and only if $\max\{i : i \in S_1 \Delta S_2\} \in S_2$.

► **Lemma 10.** The relation \prec is a strict total order.

Proof. For every subset $S \subseteq [n]$, we define $\chi(S) = \sum_{i \in S} 2^i$. Then, $S_1 \prec S_2$ if and only if $\chi(S_1) < \chi(S_2)$. Since $<$ is a total order over the integers, so is \prec over $\mathcal{P}([n])$. ◀

Using the above relation, we now show that the query function $f : [1..t_{p,k}] \rightarrow T_{p,k}$, defined by $f(I) = S_I$, can be computed in time $O(n)$.

► **Theorem 11.** Let $T_{p,k}$ be enumerated as $T_{p,k} = \{S_1, \dots, S_{t_{p,k}}\}$ where $S_1 \prec \dots \prec S_{t_{p,k}}$. Given any integer $I \in [1..t_{p,k}]$ and random access to the elements of the table t_p , the set S_I can be computed in time $O(n)$.

Proof. Algorithm 1 gives a process which starts from $t_p[n, k]$ (i.e. the total number of subsets $S \subseteq [n]$ that sum to k modulo p) and an empty set Z , and constructs S_I by going backwards ($i = n, \dots, 1$) through the rows of t_p . At the i -th step we examine $t_p[i-1, j]$ and decide whether to include i in Z or not. If we do include i then we examine another element in that row to decide a new value of I , and we also reset j .

The algorithm consists of n iterations, each of which can be performed in constant time assuming random access to the elements of t_p , and therefore the running time is $O(n)$. What is left to prove is that the output of the algorithm is indeed S_I .

■ **Algorithm 1** Fast Subset-Sum oracle.

Input: Table t_p , integers $k \in [0..p-1]$ and $I \in [1..t_{p,k}]$.
Output: The I th subset $Z \subseteq [n]$ (according to \prec) such that $\Sigma(Z) \equiv k \pmod{p}$.

- 1 Set $j = k$ and $Z = \emptyset$.
- 2 **for** $i = n, \dots, 1$ **do**
- 3 **if** $I \leq t_p[i-1, j]$ **then**
- 4 | Do nothing.
- 5 **else**
- 6 | Update $Z = Z \cup \{i\}$, $I = I - t_p[i-1, j]$ and $j = j - a_i \pmod{p}$.
- 7 **Return** Z .

Here, we provide a high level explanation of why the algorithm works, and defer a formal proof to the full version of the paper [3]. The total ordering defined by \prec implies that $T_{p,k}$ can be written as the disjoint union of two sets, $T_{p,k} = \{S_1, \dots, S_{t_p[n-1,k]}\} \cup \{S_{t_p[n-1,k]+1}, \dots, S_{t_p[n,k]}\}$, where n is not contained in any S_i in the first (left) set, and is contained in every S_i of the second (right) set. Thus, we add n to the working set Z only if $I > t_p[n-1, k]$. If this is the case, S_I is the $I - t_p[n-1, k]$ -th element of the right set. We note that removing n from each S_i in the right set gives the next bin defined over a smaller universe of size $n-1$, $\{S \subseteq [n-1] : \Sigma(S) \equiv k - a_n \pmod{p}\}$ which has $t_p[n-1, (k - a_n) \bmod p]$ elements. Therefore, by updating $n \leftarrow n-1$, $I \leftarrow I - t_p[n-1, k]$ and $k \leftarrow (k - a_n) \bmod p$ we can repeat the process to determine whether to add $n-1$ to the working set, and so on, until we reach the value 1. ◀

► **Corollary 12** (Enumerating solutions to SUBSET-SUM via dynamic programming). *Let $A = \{a_1, \dots, a_n\}$ and $T_{p,k} = \{S \subseteq [n] : \Sigma_A(S) \equiv k \pmod{p}\}$. For any $c \leq |T_{p,k}|$, it is possible to find c elements of $T_{p,k}$ in time $\tilde{O}(p+c)$.*

Proof. By Lemma 8 the table $t_p[i, j]$ can be constructed in time $O(np)$. Thereafter, by Theorem 11 each set S_I for $I \in [1..t_{p,k}]$ can be computed in time $O(n)$. ◀

An alternative method for enumerating solutions was previously known:

► **Fact 13** (Enumerating solutions to SUBSET-SUM [8]). *Let $A = \{a_1, \dots, a_n\}$ where $a_i = 2^{O(n)}$ for all i . Let $p = 2^{O(n)}$, $0 \leq k \leq p-1$, and $T_{p,k} = \{S \subseteq [n] : \Sigma_A(S) \equiv k \pmod{p}\}$. Then, for any $c \leq |T_{p,k}|$, it is possible to find c elements of $T_{p,k}$ in time $\tilde{O}(2^{n/2} + c)$.*

In comparison with Fact 13, enumerating solutions via dynamic programming is advantageous when $p < 2^{n/2}$.

3.2 Statistics about Random Bins

We describe some statistics about the distribution of the sets $T_{p,k}$ (Definition 7) when $b \in (0, 1)$ is a constant, p is a random integer in $[2^{bn}..2^{bn+1}]$, and k is a random integer in $[0..p-1]$. Therefore, in this section, we stress out that $T_{p,k}$ is a random bin and its cardinality $t_{p,k}$ is a random integer.

► **Lemma 14.** *The expected bin size can be upper bounded as $\mathbb{E}_{p,k}[t_{p,k}] \leq 2^{(1-b)n}$.*

Proof. The expected size of $T_{p,k}$ is at most $\mathbb{E}_{p,k}[t_{p,k}] \leq 2^{(1-b)n}$ since $\{T_{p,k} : 0 \leq k < p\}$ is a partition of $\mathcal{P}([n])$ with $p \geq 2^{bn}$. ◀

This result is extended to an upper-bound on the second moment of $t_{p,k}$, under the assumption that the input does not contain too many solution pairs. This bound is needed to analyze the complexity of the Variable-Time Amplitude Amplification algorithm.

► **Lemma 15.** *Fix any integer $s \geq 0$ and any real $b \in [0, 1]$. If there are at most $2^{(2-b)n}$ pairs $(S_1, S_2) \in \mathcal{P}([n])^2$ such that $\Sigma(S_1) = \Sigma(S_2) + s$, then the expected product of the sizes of two bins at distance $s \bmod p$ from each other is at most $\mathbb{E}_{p,k}[t_{p,k}t_{p,(k-s) \bmod p}] \leq \tilde{O}(2^{2(1-b)n})$.*

Proof. The expectation of $t_{p,k}t_{p,(k-s) \bmod p}$ is equal to the expected number of pairs (S_1, S_2) such that $\Sigma(S_1)$ and $\Sigma(S_2) + s$ are congruent to k modulo p , that is $\mathbb{E}_{p,k}[t_{p,k}t_{p,(k-s) \bmod p}] = \mathbb{E}_{p,k}[\sum_{S_1, S_2} \mathbb{1}_{\Sigma(S_1) \equiv_p \Sigma(S_2) + s} k]$. Since k is uniformly distributed in $[0, p-1]$, this is equal to $\sum_{S_1, S_2} \mathbb{E}_p[\frac{1}{p} \mathbb{1}_{\Sigma(S_1) \equiv_p \Sigma(S_2) + s}] \leq 2^{-bn} \sum_{S_1, S_2} \Pr_p[\Sigma(S_1) \equiv_p \Sigma(S_2) + s]$, using that $p \geq 2^{bn}$. It decomposes as $2^{-bn} (\sum_{\Sigma(S_1) = s + \Sigma(S_2)} 1 + \sum_{\Sigma(S_1) \neq s + \Sigma(S_2)} \Pr_p[\Sigma(S_1) \equiv_p \Sigma(S_2) + s])$, where the first inner term is at most $2^{(2-b)n}$ by assumption, and the second term is at most $2^{2n}n2^{-bn}$ by Fact 2. Thus, $\mathbb{E}[t_{p,k}t_{p,(k-s) \bmod p}] \leq O(2^{-bn}(2^{(2-b)n} + 2^{2n}n2^{-bn})) \leq \tilde{O}(2^{2(1-b)n})$. ◀

Finally, we provide a lower bound on the number of *distinct* subset sum values that get hashed to the random bin $T_{p,k}$.

► **Lemma 16.** *Let V be any subset of the image set $\{v \in \mathbb{N} : \exists S \subseteq [n], \Sigma(S) = v\}$. Let $v_{p,k}$ denote the number of values $v \in V$ such that $v \equiv k \pmod{p}$. Suppose that $|V| \geq 2^{(1-\ell)n}$ for some $\ell \in [0, 1]$. Then,*

$$\begin{cases} \Pr_{p,k}[v_{p,k} \geq 2^{(1-\ell-b)n-2}] = \Omega(1/n), & \text{when } \ell \leq 1-b, \\ \Pr_{p,k}[v_{p,k} \geq 1] = \Omega(\min(1/n, 2^{(1-\ell-b)n})), & \text{when } \ell > 1-b. \end{cases}$$

Proof. The expected size of $V_{p,k}$ is at least $\mathbb{E}_{p,k}[v_{p,k}] \geq |V|/p \geq |V|2^{-bn-1}$ since $\{V_{p,k} : 0 \leq k < p\}$ is a partition of V . Similarly to Lemma 15, the second moment satisfies that $\mathbb{E}_{p,k}[v_{p,k}^2] \leq O(2^{-bn}(|V| + |V|^2n2^{-bn}))$ by using Fact 2. If $\ell \leq 1-b$ we can further simplify this bound into $\mathbb{E}_{p,k}[v_{p,k}^2] \leq O(2^{-bn}|V|)$ since $|V| \geq 2^{(1-\ell)n}$ by assumption. Finally, the result is obtained by applying the Paley–Zygmund inequality $\Pr[v_{p,k} \geq \mathbb{E}[v_{p,k}]/2] \geq \frac{\mathbb{E}[v_{p,k}]^2}{4\mathbb{E}[v_{p,k}^2]}$ and the fact that $\Pr[v_{p,k} \geq 1] = \Pr[v_{p,k} > 0]$ since $v_{p,k}$ is an integer. ◀

4 Shifted-Sums

Our approach for solving SHIFTED-SUMS relies on two different quantum algorithms. In Section 4.1, we present the first algorithm (based on the representation technique), which is more involved. The description of the second algorithm (based on meet-in-the-middle) is given in Section 4.2. The running time of both algorithms, expressed in Theorems 21 and 23, are functions of the size of a *maximum solution* of the input.

► **Definition 17 (Maximum solution).** *We say that two disjoint subsets $S_1, S_2 \subseteq [n]$ that form a solution to an instance of SHIFTED-SUMS are a maximum solution if the size $|S_1| + |S_2| = \ell n$ is largest among all such solutions. We call $\ell \in (0, 1)$ the maximum solution ratio.*

By choosing the faster of these two algorithms for each $\ell \in \{1/n, 2/n, \dots, (n-1)/n\}$ until a solution has been found (or it can be concluded that no solution exists), we obtain an overall quantum algorithm for SHIFTED-SUMS the following performance:

► **Theorem 18** (SHIFTED-SUMS, quantum). *There is a quantum algorithm which, given an instance of SHIFTED-SUMS with maximum solution ratio $\ell \in (0, 1)$, outputs a solution with at least inverse polynomial probability in time $\tilde{O}(2^{\gamma(\ell)n})$ where*

$$\gamma(\ell) = \begin{cases} (1 + \ell)/4 & \text{if } \ell_1 \leq \ell \leq 3/5, & \text{(Theorem 21)} \\ \ell/2 + 1/10 & \text{if } 3/5 < \ell < \ell_2, & \text{(Theorem 21)} \\ (h(\ell) + \ell)/3 & \text{otherwise} & \text{(Theorem 23)} \end{cases}$$

and $\ell_1 \approx 0.190$ and $\ell_2 \approx 0.809$ are solutions to the equations $(h(\ell) + \ell)/3 = (1 + \ell)/4$ and $(h(\ell) + \ell)/3 = \ell/2 + 1/10$ respectively. In particular, the worst case complexity is $O(2^{0.504n})$.

Since a potential solution can be verified in polynomial time in n , in what follows we describe our algorithms on yes instances with maximum solution ratio ℓ . As presented, the algorithms find a solution with inverse polynomial probability in n , which can be amplified to constant probability in polynomial time.

4.1 Representation Technique Algorithm

Our representation technique based algorithm is given in Algorithm 2, and uses the dynamic programming table of Section 3. Before constructing that table, we first check whether the input contains many solution pairs (in which case a simple quantum search is sufficient). Depending on the value of the maximum solution ratio ℓ , we may also need to apply Variable-Time Amplitude Amplification (Fact 4) on top of Quantum Pair Finding (Theorem 6).

■ **Algorithm 2** Quantum representation technique for SHIFTED-SUMS.

Input: Instance (a, s) of SHIFTED-SUMS with $\sum_{i=1}^n a_i < 2^{4n}$ and maximum solution ratio ℓ .

Output: Two subsets $S_1, S_2 \subseteq [n]$.

- 1 Set $b = (1 + \ell)/4$ if $\ell \leq 3/5$ and $b = 2/5$ if $\ell > 3/5$.
 - 2 Run the Quantum Search algorithm (Fact 3) over the set of pairs $(S_1, S_2) \in \mathcal{P}([n])^2$, where a pair is marked if $\Sigma(S_1) = \Sigma(S_2) + s$ and $S_1 \neq S_2$. Stop it and proceed to step 3 if the running time exceeds $\tilde{O}(2^{bn/2})$, otherwise output the pair it found within the allotted time.
 - 3 If $\ell > 3/5$ then run Variable-Time Amplitude Amplification (Fact 4) on steps 4 - 6, otherwise run them once:
 - 4 Choose a random prime $p \in [2^{bn} \dots 2^{bn+1}]$ and a random integer $k \in [0 \dots p - 1]$.
 - 5 Construct the table $t_p[i, j]$ for $i = 0, \dots, n$ and $j = 0, \dots, p - 1$ (see Section 3).
 - 6 Run the quantum Pair Finding algorithm (Theorem 6) to find if there exists two sets $S_1 \in T_{p,k}$ and $S_2 \in T_{p,(k-s) \bmod p}$ such that $\Sigma(S_1) = \Sigma(S_2) + s$ and $S_1 \neq S_2$. If so, output the pair (S_1, S_2) it found.
-

Note that ‘run Variable-Time Amplitude Amplification on steps 4 - 6’ means that one should apply the procedure implicit in Fact 4 to the algorithm \mathcal{A} defined by the following process (i) create a uniform superposition over all primes $p \in [2^{bn} \dots 2^{bn+1}]$ and, for each p , all $k \in [0 \dots p - 1]$. (ii) For each p , coherently construct the table t_p . (iii) Run Quantum Pair Finding coherently on each pair of sets $T_{p,k}, T_{p,(k-s) \bmod p}$, marking the (p, k) tuple if a pair is found.

The analysis of Algorithm 2 relies on the random bins statistics presented in Section 3.2. We first define the *collision values set* which contains the values of all the possible solution pairs.

► **Definition 19** (COLLISION VALUES SET). *Given an instance (a, s) to the SHIFTED-SUMS problem, the collision values set is the set $V = \{v \in \mathbb{N} : \exists S_1 \neq S_2, v = \Sigma(S_1) = \Sigma(S_2) + s\}$.*

We show that the collision values set V is of size at least $2^{(1-\ell)n}$ when the maximum solution ratio is ℓ . Thus, by Lemma 16, we can lower bound the number of values in V that get hashed to a random bin $T_{p,k}$.

► **Lemma 20.** *If the maximum solution ratio is ℓ then $|V| \geq 2^{(1-\ell)n}$.*

Proof. Let $S_1, S_2 \subseteq \{1, \dots, n\}$ be a maximum solution of size $|S_1| + |S_2| = \ell n$. Then for any $S \subseteq [n] \setminus (S_1 \cup S_2)$ the sets $S_1 \cup S$ and $S_2 \cup S$ form a solution, and for $S \neq S'$, the values $\Sigma(S_1 \cup S)$ and $\Sigma(S_1 \cup S')$ must be distinct. If this were not the case then $S_1 \cup (S \setminus S')$ and $S_2 \cup (S' \setminus S)$ would form a disjoint solution of size larger than ℓ . ◀

We now prove the correctness of Algorithm 2.

► **Theorem 21** (SHIFTED-SUMS, representation). *Given an instance of SHIFTED-SUMS with $\sum_{i=1}^n a_i < 2^{4n}$ and maximum solution ratio $\ell \in (0, 1)$, Algorithm 2 finds a solution with inverse polynomial probability in time $\tilde{O}(2^{(1+\ell)n/4})$ if $\ell \leq 3/5$, and $\tilde{O}(2^{(\ell/2+1/10)n})$ if $\ell > 3/5$.*

Proof. Step 2 of Algorithm 2 handles the case where the total number of solution pairs exceeds $2^{(2-b)n}$. In this situation, the Quantum Search algorithm can find a solution pair in time $\tilde{O}(\sqrt{2^{2n}/2^{(2-b)n}}) = \tilde{O}(2^{bn/2})$, which is smaller than the complexity given in Theorem 21.

Analysis when $\ell \leq 3/5$. In this case the algorithm executes steps 4 - 6 only once. From Lemma 8, the table t_p can be constructed in time $O(n2^{bn})$, after which each query to the elements of $T_{p,k}$ can be performed in time $O(n)$ (Theorem 11). By Lemma 16, the number of disjoint solution pairs contained in $T_{p,k} \times T_{p,(k-s) \bmod p}$ is at least $v_{p,k} \geq 2^{(1-\ell-b)n-2}$ with probability $\Omega(1/n)$. By Lemma 14 and Markov's inequality, the sizes of $T_{p,k}$ and $T_{p,(k-s) \bmod p}$ are at most $t_{p,k}, t_{p,(k-s) \bmod p} \leq n^2 2^{(1-b)n}$ with probability at least $1 - 1/n^2$. Thus, with probability $\Omega(1/n)$ we can assume that both of these events occur. If this is the case, then the time to execute step 6 of the algorithm is $\tilde{O}\left((t_{p,k} t_{p,(k-s) \bmod p} / v_{p,k})^{1/3}\right) = \tilde{O}(2^{(1+\ell-b)n/3})$ since the first complexity given in Theorem 6 is the largest one for our choice of parameters. This is at most $\tilde{O}(2^{(1+\ell)n/4})$ when $b = (1 + \ell)/4$.

Analysis when $\ell > 3/5$. We assume that the total number of solution pairs is at most $2^{(2-b)n}$ (otherwise we would have found a collision at step 2 with high probability). Given p and k , the base algorithm (steps 4 - 6) succeeds if there is a solution in $T_{p,k} \times T_{p,(k-s) \bmod p}$, i.e. $v_{p,k} \geq 1$. Therefore by Lemmas 20 and 16, we have for its success probability $\rho = \Omega(\min(1/n, 2^{(1-\ell-b)n}))$. We claim that $\mathbb{E}[\tau^2] = \tilde{O}(2^{2bn})$ where τ is the stopping time of the base algorithm. Constructing the table t_p takes time $\tilde{O}(p)$, and by summing the two complexities given in Theorem 6 the Quantum Pair Finding algorithm takes time at most $\tilde{O}\left((t_{p,k} t_{p,(k-s) \bmod p})^{1/3} + \sqrt{\max(t_{p,k}, t_{p,(k-s) \bmod p})}\right)$. Therefore we have $\mathbb{E}[\tau^2] = \tilde{O}(\mathbb{E}_{p,k}[(p + (t_{p,k} t_{p,(k-s) \bmod p})^{1/3} + \sqrt{\max(t_{p,k}, t_{p,(k-s) \bmod p})})^2]) \leq \tilde{O}(\mathbb{E}_{p,k}[p^2] + \mathbb{E}_{p,k}[t_{p,k}^{2/3} t_{p,(k-s) \bmod p}^{2/3}] + \mathbb{E}_{p,k}[t_{p,k}] \leq \tilde{O}(\mathbb{E}_{p,k}[p^2] + \mathbb{E}_{p,k}[t_{p,k} t_{p,(k-s) \bmod p}]^{2/3} + \mathbb{E}_{p,k}[t_{p,k}]) \leq \tilde{O}(2^{2bn}) + \tilde{O}(2^{4(1-b)n/3}) + 2^{(1-b)n}$, where the second inequality uses that the moment function is non-decreasing and the last inequality uses Lemmas 14 and 15. Since $b = 2/5$ we obtain that $\mathbb{E}[\tau^2] \leq \tilde{O}(2^{2bn})$. Finally, by Fact 4, the overall time of steps 3 - 6 is $\tilde{O}(\sqrt{\mathbb{E}[\tau^2]/\rho}) = \tilde{O}(2^{bn}/2^{(1-\ell-b)n/2}) = \tilde{O}(2^{(\frac{2}{5} + \frac{1}{10})n})$. ◀

4.2 Quantum Meet-in-the-Middle Algorithm

We describe the second quantum algorithm for solving SHIFTED-SUMS based on the meet-in-the-middle technique combined with Quantum Search. We first state a lemma that if we randomly partition the input into two sets of relative sizes 1:2, then with at least inverse polynomial probability a maximum solution will be distributed in the same proportion between the two sets.

► **Lemma 22.** *Let S_1, S_2 be a maximum solution of ratio ℓ . Then with at least inverse polynomial probability the random partition $X_1 \cup X_2$ satisfies $|(S_1 \cup S_2) \cap X_1| = \ell n/3$, $|(S_1 \cup S_2) \cap X_2| = 2\ell n/3$.*

Proof. There are $\binom{n}{n/3}$ ways to partition $[n]$ into two subsets X_1 and X_2 of respective sizes $n/3$ and $2n/3$. Of these, there are $\binom{\ell n}{\ell n/3} \cdot \binom{n-\ell n}{\frac{n-\ell n}{3}}$ partitions such that $|(S_1 \cup S_2) \cap X_1| = \ell n/3$, $|(S_1 \cup S_2) \cap X_2| = 2\ell n/3$. The probability that $|(S_1 \cup S_2) \cap X_1| = \ell n/3$, $|(S_1 \cup S_2) \cap X_2| = 2\ell n/3$ is thus $\frac{\binom{\ell n}{\ell n/3} \cdot \binom{n-\ell n}{\frac{n-\ell n}{3}}}{\binom{n}{n/3}}$. Fact 1 gives that this quantity is at least $\Omega(n^{-1/2})$. ◀

We use the above result in the design of Algorithm 3, which is analyzed in the next theorem. We observe that the obtained time complexity is at most $\tilde{O}(3^{n/3})$ and it is maximized at $\ell = 2/3$.

■ **Algorithm 3** Quantum meet-in-the-middle technique for SHIFTED-SUMS.

Input: Instance (a, s) of SHIFTED-SUMS with maximum solution ratio ℓ .

Output: Two subsets $S_1, S_2 \subseteq [n]$.

- 1 Randomly split $[n]$ into disjoint subsets $X_1 \cup X_2$ such that $|X_1| = n/3, |X_2| = 2n/3$.
 - 2 Classically compute and sort $V_1 = \{\Sigma(S_{11}) - \Sigma(S_{21}) : S_{11}, S_{21} \subseteq X_1 \text{ and } S_{11} \cap S_{21} = \emptyset \text{ and } |S_{11}| + |S_{21}| = \ell n/3\}$.
 - 3 Apply Quantum Search (Fact 3) over the set $V_2 = \{\Sigma(S_{12}) - \Sigma(S_{22}) : S_{12}, S_{22} \subseteq X_2 \text{ and } S_{12} \cap S_{22} = \emptyset \text{ and } |S_{12}| + |S_{22}| = 2\ell n/3\}$, where an element $v_2 \in V_2$ is marked if there exists $v_1 \in V_1$ such that $v_1 + v_2 = s$. For a marked v_2 , output $S_1 = S_{11} \cup S_{12}$ and $S_2 = S_{21} \cup S_{22}$.
-

► **Theorem 23** (SHIFTED-SUMS, meet-in-the-middle). *Given an instance of SHIFTED-SUMS with maximum solution ratio $\ell \in (0, 1)$, Algorithm 3 finds a solution with at least inverse polynomial probability in time $\tilde{O}(2^{n(h(\ell)+\ell)/3})$.*

Proof. There are $\binom{n/3}{\ell n/3} 2^{\ell n/3}$ different ways to select two sets $S_{11}, S_{21} \subseteq X_1$ such that $S_{11} \cap S_{21} = \emptyset$, $|S_{11}| + |S_{21}| = \ell n/3$. Computing and sorting V_1 thus takes time $\tilde{O}(\binom{n/3}{\ell n/3} 2^{\ell n/3})$. In the next step of the algorithm, Quantum Search is performed over all $\binom{2n/3}{2\ell n/3} 2^{2\ell n/3}$ sets $S_{12}, S_{22} \subseteq X_2$ such that $S_{12} \cap S_{22} = \emptyset$, $|S_{12}| + |S_{22}| = 2\ell n/3$. We mark an element $v_2 \in V_2$ if there exists $v_1 \in V_1$ such that $v_1 + v_2 = s$. Since V_1 is sorted this check can be done in time $\text{polylog}(|V_1|)$. The total time required is therefore $\tilde{O}\left(\binom{n/3}{\ell n/3} 2^{\ell n/3} + \sqrt{\binom{2n/3}{2\ell n/3} 2^{2\ell n/3}}\right) = \tilde{O}\left(2^{\ell n/3} \left(\binom{n/3}{\ell n/3} + \sqrt{\binom{2n/3}{2\ell n/3}}\right)\right) = \tilde{O}\left(2^{\frac{2}{3}(h(\ell)+\ell)n}\right)$. By Lemma 22, when the instance has a maximum solution of size ℓn , the set V_2 has a marked element with at least inverse polynomial probability, and in that case a solution is found. ◀

5 Pigeonhole Equal-Sums

We give classical and quantum algorithms for the PIGEONHOLE EQUAL-SUMS problem, based on dynamic programming and which run in time $\tilde{O}(2^{n/2})$ and $\tilde{O}(2^{2n/5})$, respectively. In contrast with our quantum algorithm for SHIFTED-SUMS which made use of a random prime modulus, in the case of PIGEONHOLE EQUAL-SUMS we can deterministically choose a modulus p , and the pigeonhole principle guarantees a collision in at least one bin.

► **Lemma 24.** *There is a classical deterministic algorithm such that, given an instance of PIGEONHOLE EQUAL-SUMS and a modulus p that divides 2^n , it finds in time $\tilde{O}(p)$ an integer k such that there exists two distinct subsets S_1, S_2 with $\Sigma(S_1) \equiv \Sigma(S_2) \equiv k \pmod{p}$.*

Proof. Denote by $\bar{0}, \bar{1}, \dots, \overline{p-1}$ the congruence classes modulo p . Each of these classes contains exactly $2^n/p$ numbers between 0 and $2^n - 2$, except the last class $\overline{p-1}$ that has only $2^n/p - 1$ numbers. Since all 2^n subsets $S \subseteq [n]$ have a sum $\Sigma(S)$ between 0 and $2^n - 2$ there are two possible cases:

- either there is some class \bar{k} such that $\Sigma(S) \in \bar{k}$ for strictly more than $2^n/p$ subsets S ,
- or there are $2^n/p$ subsets S such that $\Sigma(S) \in \overline{p-1}$.

Denote by \bar{k} a class that verifies one of these two points. By definition, there are *strictly more* subsets S such that $\Sigma(S) \in \bar{k}$ than the number of elements between 0 and $2^n - 2$ that belong to \bar{k} . However, for all $S \subseteq [n]$, we have $\Sigma(S) \leq 2^n - 2$. Thus, there must be two subsets $S_1 \neq S_2$ such that $\Sigma(S_1), \Sigma(S_2) \in \bar{k}$ and $\Sigma(S_1) = \Sigma(S_2)$.

From Lemma 8, the table $t_p[i, j] = |\{S \subseteq \{1, \dots, i\} : \Sigma(S) \equiv j \pmod{p}\}|$ can be constructed in time $\tilde{O}(p)$. From the table, we can read off a value k that satisfies the above condition. ◀

► **Theorem 25** (PIGEONHOLE EQUAL-SUMS, classical). *There is a classical deterministic algorithm for the PIGEONHOLE EQUAL-SUMS problem that runs in time $\tilde{O}(2^{n/2})$.*

Proof. Choose $p = 2^{n/2}$. By Lemma 24, in time $\tilde{O}(2^{n/2})$ we can find k such that there exists $S_1 \neq S_2$ satisfying $\Sigma(S_1) \equiv \Sigma(S_2) \equiv k \pmod{2^{n/2}}$. Once we know a bin that contains a collision, by Corollary 12, we can enumerate in time $\tilde{O}(2^{n/2})$ a sufficient number of subsets in that bin to locate a collision. ◀

► **Theorem 26** (PIGEONHOLE EQUAL-SUMS, quantum). *There is a quantum algorithm for the PIGEONHOLE EQUAL-SUMS problem that runs in time $\tilde{O}(2^{2n/5})$.*

Proof. We set $p = 2^{2n/5}$ and, by Lemma 24, in time $\tilde{O}(2^{2n/5})$ we can identify k such that there exists $S_1 \neq S_2$ satisfying $\Sigma(S_1) \equiv \Sigma(S_2) \equiv k \pmod{2^{2n/5}}$. By Theorem 11, each query to $T_{p,k} = \{S \subseteq [n] : \Sigma(S) \equiv k \pmod{p}\}$ can be made in time $O(n)$. We use Ambainis' element distinctness algorithm [4] on these elements to find a collision. We do not want to run it on an unnecessarily large set. Therefore, if $t_{p,k} > 2^{3n/5+1}$ then we run it only on the first $2^{3n/5+1}$ elements of $T_{p,k}$, according to the ordering defined by \prec . A collision is then found in time $\tilde{O}((2^{3n/5})^{2/3}) = \tilde{O}(2^{2n/5})$. The overall running time of the algorithm is thus $\tilde{O}(2^{2n/5})$. ◀

In the full version [3], we give an $\tilde{O}(2^{n/2})$ classical deterministic algorithm for the following modular version of PIGEONHOLE EQUAL-SUMS. We also ask the open question of finding a faster quantum algorithm.

► **Problem 6** (PIGEONHOLE MODULAR EQUAL-SUMS). Given a set $\{a_1, \dots, a_n\}$ of positive integers and a modulus q such that $q \leq 2^n - 1$, find two distinct subsets $S_1, S_2 \subseteq [n]$ such that $\sum_{i \in S_1} a_i \equiv \sum_{i \in S_2} a_i \pmod{q}$.

References

- 1 A. Abboud. Fine-grained reductions and quantum speedups for dynamic programming. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 8:1–8:13, 2019.
- 2 A. Abboud, K. Bringmann, D. Hermelin, and D. Shabtay. SETH-based lower bounds for subset sum and bicriteria path. In *Proceedings of the 30th Symposium on Discrete Algorithms (SODA)*, pages 41–57, 2019.
- 3 J. Allcock, Y. Hamoudi, A. Joux, F. Klingelhöfer, and M. Santha. Classical and quantum algorithms for variants of subset-sum via dynamic programming, 2021. [arXiv:2111.07059 \[quant-ph\]](https://arxiv.org/abs/2111.07059).
- 4 A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.
- 5 A. Ambainis. Variable time amplitude amplification and quantum algorithms for linear algebra problems. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 636–647, 2012.
- 6 A. Ambainis, K. Balodis, J. Iraids, M. Kokainis, K. Prusis, and J. Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the 30th Symposium on Discrete Algorithms (SODA)*, pages 1783–1793, 2019.
- 7 C. Bazgan, M. Santha, and Z. Tuza. Efficient approximation algorithms for the Subset-Sums Equality problem. *Journal of Computer and System Sciences*, 64(2):160–170, 2002.
- 8 A. Becker, J.-S. Coron, and A. Joux. Improved generic algorithms for hard knapsacks. In *Proceedings of the 30th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 364–385, 2011.
- 9 D. J. Bernstein, S. Jeffery, T. Lange, and A. Meurer. Quantum algorithms for the subset-sum problem. In *Proceedings of the 5th International Workshop on Post-Quantum Cryptography (PQCrypto)*, pages 16–33, 2013.
- 10 X. Bonnetain, R. Bricout, A. Schrottenloher, and Y. Shen. Improved classical and quantum algorithms for subset-sum. In *Proceedings of the 26th International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT)*, pages 633–666, 2020.
- 11 M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.
- 12 K. Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th Symposium on Discrete Algorithms (SODA)*, pages 1073–1084, 2017.
- 13 H. Buhrman, B. Loff, and L. Torenvliet. Hardness of approximation for knapsack problems. *Theory of Computing Systems*, 56(2):372–393, 2015.
- 14 X. Chen, Y. Jin, T. Randolph, and R. A. Servedio. Average-case subset balancing problems. In *Proceedings of the 33rd Symposium on Discrete Algorithms (SODA)*, pages 743–778, 2022.
- 15 A. Esser and A. May. Low weight discrete logarithm and subset sum in $2^{0.65n}$ with polynomial memory. In *Proceedings of the 39th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 94–122, 2020.
- 16 R. G. Gallager. *Information Theory and Reliable Communication*. John Wiley and Sons, 1968.
- 17 G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford, fourth edition, 1975.
- 18 A. Helm and A. May. Subset sum quantumly in 1.17^n . In *Proceedings of the 13th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC)*, pages 5:1–5:15, 2018.
- 19 E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21(2):277–292, 1974.
- 20 N. Howgrave-Graham and A. Joux. New generic algorithms for hard knapsacks. In *Proceedings of the 29th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 235–256, 2010.

- 21 K. Jansen, F. Land, and K. Land. Bounding the running time of algorithms for scheduling and packing problems. *SIAM Journal on Discrete Mathematics*, 30(1):343–366, 2016.
- 22 S. Jeffery. *Frameworks for Quantum Algorithms*. PhD thesis, University of Waterloo, 2014.
- 23 R. M. Karp. Reducibility among combinatorial problems. In *Proceedings of the Symposium on the Complexity of Computer Computations*, pages 85–103, 1972.
- 24 H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.
- 25 F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.
- 26 N. Megiddo and C. H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.
- 27 M. Mucha, J. Nederlof, J. Pawlewicz, and K. Wegrzycki. Equal-subset-sum faster than the meet-in-the-middle. In *Proceedings of the 27th European Symposium on Algorithms (ESA)*, pages 73:1–73:16, 2019.
- 28 C. H. Papadimitriou. On graph-theoretic lemmata and complexity classes (extended abstract). In *Proceedings of the 31st Symposium on Foundations of Computer Science (FOCS)*, pages 794–801, 1990.
- 29 K. Sotiraki, M. Zampetakis, and G. Zirdelis. Ppp-completeness with connections to cryptography. In *Proceedings of the 59th Symposium on Foundations of Computer Science (FOCS)*, pages 148–158, 2018.
- 30 S. Tani. Claw finding algorithms using quantum walk. *Theoretical Computer Science*, 410(50):5285–5297, 2009.
- 31 G. J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.
- 32 G. J. Woeginger and Z. Yu. On the equal-subset-sum problem. *Information Processing Letters*, 42(6):299–302, 1992.

Techniques for Generalized Colorful k -Center Problems

Georg Anegg  

ETH Zürich, Switzerland

Laura Vargas Koch  

ETH Zürich, Switzerland

University of Chile, Santiago, Chile

Rico Zenklusen  

ETH Zürich, Switzerland

Abstract

Fair clustering enjoyed a surge of interest recently. One appealing way of integrating fairness aspects into classical clustering problems is by introducing multiple covering constraints. This is a natural generalization of the robust (or outlier) setting, which has been studied extensively and is amenable to a variety of classic algorithmic techniques. In contrast, for the case of multiple covering constraints (the so-called colorful setting), specialized techniques have only been developed recently for k -Center clustering variants, which is also the focus of this paper.

While prior techniques assume covering constraints on the clients, they do not address additional constraints on the facilities, which has been extensively studied in non-colorful settings. In this paper, we present a quite versatile framework to deal with various constraints on the facilities in the colorful setting, by combining ideas from the iterative greedy procedure for Colorful k -Center by Inamdar and Varadarajan with new ingredients. To exemplify our framework, we show how it leads, for a constant number γ of colors, to the first constant-factor approximations for both Colorful Matroid Supplier with respect to a linear matroid and Colorful Knapsack Supplier. In both cases, we readily get an $O(2^\gamma)$ -approximation.

Moreover, for Colorful Knapsack Supplier, we show that it is possible to obtain constant approximation guarantees that are independent of the number of colors γ , as long as $\gamma = O(1)$, which is needed to obtain a polynomial running time. More precisely, we obtain a 7-approximation by extending a technique recently introduced by Jia, Sheth, and Svensson for Colorful k -Center.

2012 ACM Subject Classification Theory of computation \rightarrow Facility location and clustering

Keywords and phrases Approximation Algorithms, Fair Clustering, Colorful k -Center

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.7

Related Version *Full Version:* <http://arxiv.org/abs/2207.02609>

Funding *Georg Anegg:* Research supported in part by Swiss National Science Foundation grant number 200021_184622.



Rico Zenklusen: Research supported in part by Swiss National Science Foundation grant number 200021_184622. 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 817750).



1 Introduction

As more and more decisions are automated, there has been an increasing interest in incorporating fairness aspects in algorithms by design. This applies in particular to clustering problems, where considerable attention has recently been dedicated to developing and studying various models of fair clustering, see, e.g., [8], [3], and [2].



© Georg Anegg, Laura Vargas Koch, and Rico Zenklusen;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 7; pp. 7:1–7:14

Leibniz International Proceedings in Informatics



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

In this paper, we focus on the so-called colorful setting, which was introduced in [3]. In colorful clustering, each client is a member of certain subgroups and every clustering is required to cover at least a given number of clients of each subgroup. This may be considered under various clustering objectives (like k -median and k -mean), though only the k -center case has been studied so far.

Colorful clustering is an appealing notion as it is a natural generalization of the robust (or outlier) setting, where there is only a single group which every client belongs to. Various clustering problems have been studied in depth in the robust setting, see, e.g., [5], [9], and [2].

While the robust setting is amenable to a variety of well-known and basic algorithmic techniques, the only constant-factor approximations for the colorful setting, which imposes multiple covering constraints leading to more balanced clusterings, are based on significantly more sophisticated techniques, tailored specifically to those settings. More precisely, three distinct techniques have been successful at achieving constant-factor approximations in the context of colorful k -center clustering, namely the combinatorial approach of [11], the round-or-cut-based approach of [1], and the iterative greedy reductions of [10].

However, these approaches do not immediately generalize to variants with constraints on the facilities, even for the common Matroid Center or Knapsack Center clustering variants. On the other hand, techniques for the Knapsack and Matroid k -Center problems in the robust setting (see [5] and [9]) do not easily extend to multiple covering constraints.

Thus, prior to this work, no approaches have been known that lead to constant-factor approximations for colorful variants of otherwise well-studied k -center problems like Matroid Center or Knapsack Center. Filling this gap is the goal of this paper.

1.1 Our contributions

Our main contribution is a partitioning procedure which leads to a general reduction of colorful k -center clustering problems with constraints on the facilities to a significantly simpler multi-dimensional covering problem (see Theorem 3). This reduction comes at the cost of a constant factor depending on the number of colors.

It is inspired by recent insights of [10] on decoupling multiple covering constraints and iteratively applying a greedy partitioning procedure of [6]. By taking into account multiple colors at the same time, our framework gives an improved way of dealing with multiple covering constraints while also becoming more versatile. Our framework also extends and simplifies ideas of the approximation algorithm for Robust Matroid Center of [7].

We start by introducing the γ -Colorful \mathcal{F} -Supplier problem, which formalizes colorful k -center problems with (down-closed) constraints on the facilities.

► **Definition 1** (γ -Colorful \mathcal{F} -Supplier problem). *Let $(C \dot{\cup} F, d)$ be a finite metric space on a set of clients C and facilities F , let $\mathcal{F} \subseteq 2^F$ be a down-closed family of subsets of F , and let $\gamma \in \mathbb{Z}_{\geq 0}$. Moreover, we are given for each $\ell \in [\gamma]$:*

- *a unary encoded weight/color function $w_\ell : C \rightarrow \mathbb{Z}_{\geq 0}$, and*
- *a covering requirement $m_\ell \in \mathbb{Z}_{\geq 0}$.*

The γ -Colorful \mathcal{F} -Supplier problem asks to find the smallest radius r together with a set $S \subseteq \mathcal{F}$ such that $w_\ell(B_C(S, r)) \geq m_\ell$ for all $\ell \in [\gamma]$.¹

¹ We use the common notation $w(T) := \sum_{t \in T} w(t)$ for functions $w : U \rightarrow \mathbb{R}_{\geq 0}$ and $T \subseteq U$, as well as $B(q, r) := \{v \in C \cup F \mid d(q, v) \leq r\}$ for the ball of radius r around point q . Moreover, we use the shorthand $B_U(V, r) := \{U \cap \bigcup_{v \in V} B(v, r)\}$ for sets $U, V \subseteq C \cup F$.

We note that it is also common to define colorful k -center versions in an unweighted way (thus not using weight functions w_ℓ) by assigning to each client a subset of the γ many colors and requiring that, for each color, m_ℓ many clients of that color are covered. The definition we use clearly captures this case (and can easily be seen to be equivalent). This connection also explains why the weights w_ℓ are assumed to be given in unary encoding.

Following common terminology in the literature, when \mathcal{F} is the family of independent sets of a matroid or feasible sets with respect to a knapsack constraint, we call the problem γ -Colorful Matroid Supplier and γ -Colorful Knapsack Supplier, respectively.

Our main contribution is a general reduction of γ -Colorful \mathcal{F} -Supplier to an auxiliary problem, which we call \mathcal{F} -COVER-PROMISE (\mathcal{F} -CP). \mathcal{F} -CP, which is formally defined below, is a multi-dimensional cover problem with the added promise that highly structured solutions exist. The promise is key, as the problem without the promise can be thought of as a multi-dimensional max-cover problem.

► **Definition 2** (\mathcal{F} -COVER-PROMISE (\mathcal{F} -CP)). *In the \mathcal{F} -COVER-PROMISE problem (\mathcal{F} -CP), we are given a set family $\mathcal{H} \subseteq 2^{\mathcal{U}}$ over a finite universe \mathcal{U} , a family $\mathcal{F} \subseteq 2^{\mathcal{H}}$ of feasible subsets of \mathcal{H} , and γ many unary encoded weight functions $w_1, \dots, w_\gamma : \mathcal{U} \rightarrow \mathbb{R}_{\geq 0}$ each with a requirement m_ℓ (for $\ell \in [\gamma]$). The task is to find a feasible family of sets $S \in \mathcal{F}$ such that*

$$w_\ell \left(\bigcup_{H \in S} H \right) \geq m_\ell \quad \forall \ell \in [\gamma] .$$

The promise is that there exists a family $S \subseteq \mathcal{F}$ and a way to pick for each $H \in S$ a single representative $u_H \in H$ such that

$$w_\ell (\{u_H : H \in S\}) \geq m_\ell \quad \forall \ell \in [\gamma] .$$

In words, the promise is that there is a solution that picks a family of sets and the requirements can be fulfilled by only using a single representative u_H in each set. However, the solution we are allowed to build is such that the weight of all elements covered by our sets are counted instead of just a single representative per set.

We are now ready to state our main reduction theorem, which, as we discuss later, readily leads, for a constant number of colors γ , to the first constant-factor approximations for γ -Colorful Matroid Supplier for linear matroids and γ -Colorful Knapsack Supplier. Our reduction to \mathcal{F} -CP comes at the cost of an $O(2^\gamma)$ -factor in the approximation guarantee.

► **Theorem 3.** *For any family of down-closed set systems, we have that if \mathcal{F} -CP can be solved efficiently for any \mathcal{F} in that family, then there is an $O(2^\gamma)$ -approximation algorithm for γ -Colorful \mathcal{F} -Supplier for any \mathcal{F} in the family.²*

While the dependence of the approximation factor on γ may be undesirable, the algorithmic barriers for prior approaches remain even when $\gamma = 2$ and, for hardness reasons, we do not expect approximation algorithms to exist at all when γ grows too quickly. In particular, [1] showed that even a simple version of colorful clustering, where any k centers can be chosen, does not admit an $O(1)$ -approximation algorithm when $\gamma = \omega(\log |C \dot{\cup} F|)$ under the Exponential Time Hypothesis. Thus, in what follows, we restrict ourselves to $\gamma = O(1)$.

² When talking about the same set system \mathcal{F} both in the context of \mathcal{F} -CP and γ -Colorful \mathcal{F} -Supplier, we consider \mathcal{F} to be the same set system in both settings even if the ground sets are different, as long as there is a one-to-one relation between the ground sets mapping sets of one system to sets of the other one and vice versa.

We now discuss implications of Theorem 3 to γ -Colorful Matroid Supplier for linear matroids and γ -Colorful Knapsack Supplier. When \mathcal{F} is the family of independent sets of a linear matroid, we show how \mathcal{F} -CP can be solved with techniques relying on an efficient randomized procedure for the *Exact Weight Basis* (XWB) problem for linear matroids.³ Linear matroids include as special cases many other well-known matroid classes, including uniform matroids, and more generally partition and laminar matroids, graphic matroids, transversal matroids, gammoids, and regular matroids.

► **Theorem 4.** *For $\gamma = O(1)$ and \mathcal{F} being the independent sets of a linear matroid, \mathcal{F} -CP can be solved efficiently by a randomized algorithm. Hence (by Theorem 3), there is a randomized $O(2^\gamma)$ -approximation algorithm for γ -Colorful Matroid Supplier for linear matroids.*

The restriction to linear matroids and the fact that the algorithm is randomized are not artifacts of our framework. Indeed, by an observation in [11], rephrased for matroids below, we do not only have that XWB implies results for γ -Colorful Matroid Supplier (which will follow from our reduction), but also a reverse implication. More precisely, even for 2-Colorful Matroid Supplier, deciding whether there is a solution of radius zero requires being able to solve XWB on that matroid. However, it is unknown whether XWB can be solved efficiently on general matroids, and the only technique known for XWB on linear matroids is inherently randomized [4]. (Derandomization is a long-standing open question in this context.)

► **Lemma 5** (based on [11]). *If there is an efficient algorithm for deciding whether 2-Colorful Matroid Supplier with respect to a given class of matroids admits a solution of radius zero, then XWB can be solved efficiently on the same class of matroids.*

Note that if we cannot decide the existence of a radius zero solution, then no approximation algorithm with any finite approximation guarantee can exist.

For the case where \mathcal{F} are the feasible sets for a knapsack problem, one can use standard dynamic programming techniques to see that \mathcal{F} -CP can be solved efficiently, which readily leads to a $O(2^\gamma)$ -approximation for γ -Colorful Knapsack Supplier.

Whereas our reduction given by Theorem 3 is broadly applicable and readily leads to first constant-factor approximations for γ -Colorful \mathcal{F} -Supplier problems, it remains open whether and in which settings a dependence of the approximation factor on the number of colors is necessary. We make first progress toward this question for γ -Colorful Knapsack Supplier, where we show how techniques from [11] can be modified and extended to give a 7-approximation (independent of the number of colors).

► **Theorem 6.** *For $\gamma = O(1)$, there is a 7-approximation algorithm for γ -Colorful Knapsack Supplier.*

Our technical contribution here lies in handling the knapsack constraint in this approach – modifying the algorithm of [11] to the supplier setting and to weighted instances is straightforward. In fact, their algorithm can be seen to give a 3-approximation even for γ -Colorful k -Supplier, which is tight in light of a hardness result in [6], namely that it is NP-hard to approximate Robust k -center with forbidden centers to within $3 - \epsilon$. This remains the strongest hardness result even for γ -Colorful \mathcal{F} -Supplier problems.

³ In XWB, one is given a matroid on a ground set with unary encoded weights and a target weight; the goal is to find a basis of the matroid of weight equal to the target weight. The technique in [4] to solve XWB for linear matroids needs an explicit linear representation of the linear matroid. We make the common assumption that this is the case whenever we make a statement about linear matroids.

1.2 Organization of this paper

Our main reduction, Theorem 3, is based on what we call (L, r) -partitions, which is a way to judiciously partition the clients into parts that we want to cover together. We introduce (L, r) -partitions in Section 2 and show how the existence of certain strong (L, r) -partitions implies Theorem 3. In Section 3, we show how our reduction framework can be used to obtain first constant-factor approximations for γ -Colorful Matroid Supplier for linear matroids (thus showing Theorem 4) and γ -Colorful Knapsack Supplier. Finally, in Section 4 we prove existence of strong (L, r) -partitions. The proof of Lemma 5 as well as our 7-approximation for γ -Colorful Knapsack Supplier, i.e., the proof of Theorem 6, can be found in the extended version of this paper.

2 Reducing to \mathcal{F} -CP through (L, r) -partitions

Consider a γ -Colorful \mathcal{F} -Supplier problem on a metric space $(X = (C \dot{\cup} F), d)$ with weights $w_\ell: C \rightarrow \mathbb{Z}_{\geq 0}$ for $\ell \in [\gamma]$ and covering requirements $m_\ell \in \mathbb{Z}_{\geq 0}$ for $\ell \in [\gamma]$. An (L, r) -partition is a partition of the clients into parts of small diameter each of which we consider in our analysis to be either fully covered or not covered at all. The key property of an (L, r) -partition is that, if our instance admits a radius- r solution, then there is a radius- $(L + 1)r$ solution where we allow each center to cover only *a single part* of the partition. It is the existence of such highly structured solutions that we exploit to design $O(1)$ -approximation algorithms.

A crucial property of (L, r) -partitions is that they neither depend on \mathcal{F} nor the covering requirements m_ℓ , but only on the metric space and the weight functions, which we call a γ -colorful space for convenience.

- **Definition 7** (γ -colorful space (X, d, w)). A γ -colorful space $(X = C \dot{\cup} F, d, w)$ consists of
1. a metric space (X, d) , and
 2. color functions $w_\ell: C \rightarrow \mathbb{R}_{\geq 0}$ for $\ell \in [\gamma]$.

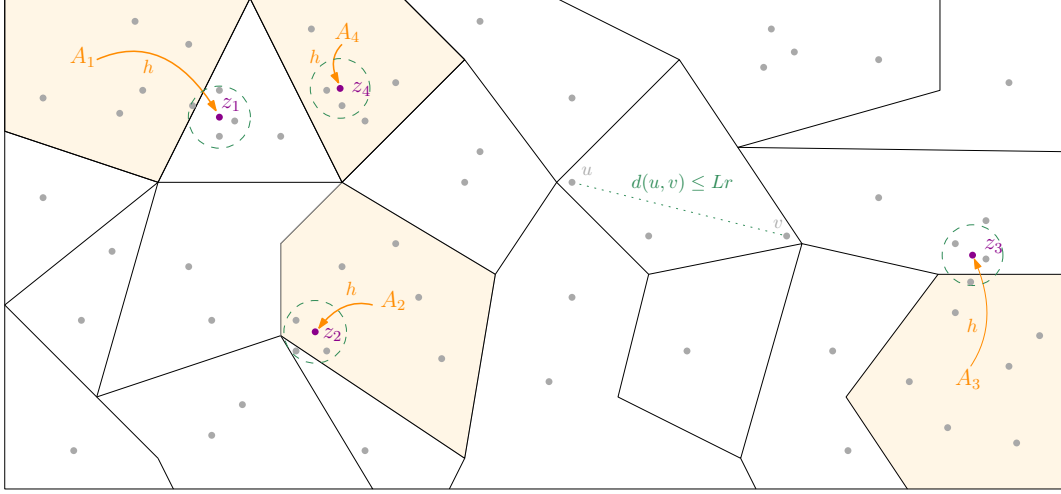
We assume for convenience that the supports of the color functions, i.e., $\text{supp}(w_\ell)$ for $\ell \in [\gamma]$, are pairwise disjoint. One can reduce to this case without loss of generality by co-locating copies of clients. We are now ready to formally define the notion of (L, r) -partition.

- **Definition 8** ((L, r) -partition). Let $(X = C \dot{\cup} F, d, w)$ be a γ -colorful space and $r, L \in \mathbb{R}_{\geq 0}$. A partition $\mathcal{P} \subseteq 2^C$ is an (L, r) -partition if

1. $\text{diam}(A) := \max_{u, v \in A} d(u, v) \leq L \cdot r \quad \forall A \in \mathcal{P}$, and
2. for any $Z \subseteq F$, there exists a subfamily $\mathcal{A} \subseteq \mathcal{P}$ and injection $h: \mathcal{A} \rightarrow Z$ such that
 - a. $d(A, h(A)) \leq r$,⁴ and
 - b. $w_\ell(\bigcup_{A \in \mathcal{A}} A) \geq w_\ell(B_C(Z, r)) \quad \forall \ell \in [\gamma]$.

To connect (L, r) -partitions to colorful clustering problems, think of $Z \in \mathcal{F}$ as centers of a γ -Colorful \mathcal{F} -Supplier problem that satisfy the covering requirements with radius r . The definition of an (L, r) -partition \mathcal{P} then implies that there is a subset $\mathcal{A} \subseteq \mathcal{P}$ of the parts such that (i) for each $A \in \mathcal{A}$ there exists an element $h(A) \in Z$ such that any client in A has distance at most $(L + 1) \cdot r$ from $h(A)$, which follows from property 1 and 2a of the definition, and (ii) the clients in \mathcal{A} cover as much as $B_C(Z, r)$ in each color. Thus, the set of facilities $h(\mathcal{A})$ satisfies the covering requirements with respect to the radius $(L + 1) \cdot r$, and,

⁴ For any set $V \subseteq F \dot{\cup} C$ and $x \in F \dot{\cup} C$, we use the shorthand $d(V, x) := \min\{d(v, x) : v \in V\}$.



■ **Figure 1** Illustration of an (L, r) -partition of a 1-colorful space (where all points have unit weight). For $Z = \{z_i \mid i \in [4]\}$, the mapping h maps A_i to z_i for $i \in [4]$. Note that $\cup_{i \in [4]} A_i$ contains at least as many points than $\cup_{i \in [4]} B(r, z_i)$ and that $d(z_i, A_i) \leq r$ for $i \in [4]$. Furthermore, the largest distance between any two points in a set A_i is bounded by Lr .

furthermore, $h(\mathcal{A})$ is feasible because $h(\mathcal{A}) \subseteq Z$ and \mathcal{F} is down-closed. In short, $h(\mathcal{A})$ is an $(L + 1)$ -approximate solution to the γ -Colorful \mathcal{F} -Supplier problem. Hence, to obtain an $(L + 1)$ -approximation, the problem reduces to deciding which of the parts of \mathcal{P} to cover. A key simplification we gain from this connection is that the client sets in \mathcal{P} are non-overlapping because \mathcal{P} is a partition, which we will heavily exploit later to design our algorithms.

The key structural result of our work is to show that (L, r) -partitions with constant L (for a fixed γ) exist and can also be constructed efficiently, which is summarized below.

► **Lemma 9.** *For every γ -colorful space (X, d, w) and $r \in \mathbb{R}_{\geq 0}$, one can construct in polynomial time a $(10(2^\gamma - 1), r)$ -partition.⁵*

We defer the proof of Lemma 9 to Section 4, and first show how it implies our main reduction theorem, Theorem 3, and how this reduction readily leads to $O(1)$ -approximations for γ -Colorful Matroid Supplier for linear matroids and γ -Colorful Knapsack Supplier.

Proof of Theorem 3. Consider an instance of the γ -Colorful \mathcal{F} -Supplier Problem on a γ -colorful space (X, d, w) . We can guess the radius r of an optimal solution to the problem. This can be achieved by considering all pairwise distances between facilities F and clients C , repeating the steps below for each guess and only considering the best output (and discarding outputs where the procedure fails). Hence, assume that r is the optimal radius from now on.

By Lemma 9, we can efficiently construct an (L, r) -partition \mathcal{P} of (X, d, w) for $L = 10(2^\gamma - 1) = O(2^\gamma)$. Consider the \mathcal{F} -CP instance with universe $\mathcal{U} := \mathcal{P}$, family of sets

$$\mathcal{H} := \{H_f : f \in F\} \quad , \text{ where}$$

$$H_f := \{A \in \mathcal{P} \text{ with } d(A, f) \leq r\} \quad \forall f \in F \quad .$$

⁵ As we highlight later, a more careful analysis of our approach allows for a slight improvement in the constant factor, leading to the construction of $(8 \cdot 2^\gamma - 10, r)$ -partitions. However, in the interest of simplicity, we present a simpler analysis that shows the bound claimed in the lemma.

The family of feasible subsets of \mathcal{H} is the same as \mathcal{F} when identifying H_f with the element f . To make this relation explicit, if we denote by $\mathcal{F}_{\mathcal{H}}$ the family of feasible subsets, then some subset of \mathcal{H} , say $\{H_f: f \in I\}$ where $I \subseteq F$, is in $\mathcal{F}_{\mathcal{H}}$ if and only if $I \in \mathcal{F}$. Moreover, the weights and coverage thresholds are inherited from those of the given γ -Colorful \mathcal{F} -Supplier problem; formally, for $\ell \in [\gamma]$, the ℓ -th weight of $A \in \mathcal{U}$ is given by $w_\ell(A)$.

To make sure that this indeed leads to an \mathcal{F} -CP problem, we have to verify that the promise holds. Thus, let $Z \subseteq F$ be a solution to the given γ -Colorful \mathcal{F} -Supplier problem for radius r , which exists because we assume that r was guessed correctly. As \mathcal{P} is an (L, r) -partition of (X, d, w) , there is a subfamily $\mathcal{A} \subseteq \mathcal{P}$ and injection $h: \mathcal{A} \rightarrow Z$ satisfying property 2 of Definition 8. We claim that a solution fulfilling the promise is given by choosing

$$S = \{H_f: f \in h(\mathcal{A})\} \in \mathcal{F}_{\mathcal{H}} ,$$

and setting as representative element $u_{H_f} \in H_f$ the element $u_{H_f} = A_f$, where $A_f = h^{-1}(f)$. Note that because $h(\mathcal{A}) \subseteq Z \in \mathcal{F}$ and \mathcal{F} is down-closed, we indeed have $S \in \mathcal{F}_{\mathcal{H}}$. Furthermore, because the injection h satisfies $d(A_f, h(A_f)) \leq r$, we have $u_{H_f} \in H_f$, as desired. Moreover,

$$\sum_{f \in h(\mathcal{A})} w_\ell(u_{H_f}) = \sum_{f \in h(\mathcal{A})} w_\ell(A_f) = \sum_{A \in \mathcal{A}} w_\ell(A) \geq w_\ell(B_C(Z, r)) \geq m_\ell \quad \forall \ell \in [\gamma] ,$$

where the first inequality follows because \mathcal{A} fulfills the second property of Definition 8, and the last inequality is a consequence of Z being centers that are a radius- r solution to the given γ -Colorful \mathcal{F} -Supplier problem. Hence, the promised solution exists.

Thus, we can compute an \mathcal{F} -CP solution $S_{\mathcal{H}} \subseteq \mathcal{F}_{\mathcal{H}}$, which can be written as $S_{\mathcal{H}} := \{H_f: f \in S\}$ for some $S \in \mathcal{F}$. We claim that S is a solution to the given γ -Colorful \mathcal{F} -Supplier problem with radius $(L+1) \cdot r$, which finishes the proof. This follows from the fact that $S_{\mathcal{H}}$ is an \mathcal{F} -CP solution, and that, for any $f \in F$, each client in $\bigcup_{A \in H_f} A$ has distance at most $(L+1) \cdot r$ from f because \mathcal{P} is an (L, r) -partition. Hence, the clustering solution with centers S and radius $(L+1) \cdot r$ covers all clients in

$$\bigcup_{f \in S} \bigcup_{A \in H_f} A ,$$

and the w_ℓ -weight (for any $\ell \in [\gamma]$) that it covers is at least

$$w_\ell \left(\bigcup_{f \in S} \bigcup_{A \in H_f} A \right) = \sum_{A \in \bigcup_{f \in S} H_f} w_\ell(A) \geq m_\ell ,$$

where the equality uses that the ground set $\mathcal{U} = \mathcal{P}$ consists of sets A that are disjoint, and the inequality holds because $S_{\mathcal{H}} = \{H_f: f \in S\}$ is a solution to \mathcal{F} -CP. Thus, all coverage requirements are fulfilled by the clustering with centers S and radius $(L+1) \cdot r$, as desired. \blacktriangleleft

3 Applications of our reduction framework

We now discuss implications of our reduction framework, Theorem 3, to γ -Colorful Matroid Supplier for linear matroids and γ -Colorful Knapsack Supplier.

3.1 γ -Colorful Matroid Supplier

To apply our reduction framework to γ -Colorful Matroid Supplier for linear matroids, we have to solve \mathcal{F} -CP when \mathcal{F} are the independent sets of a linear matroid. We show how this problem can be reduced to XWB in a suitably defined matroid. More precisely, we use a

7:8 Techniques for Generalized Colorful k -Center Problems

reduction to the *Exact Weight Independent Set* (XWI) problem for matroids. This problem is identical to XWB except that an *independent set* with the desired target weight needs to be returned, instead of a basis. However, XWI easily reduces to XWB on linear matroids, by adding zero weight copies of the elements.

This reduction relies on Rado matroids, which is a way to construct a matroid from another one (see, e.g., [13, Section 8.2]).⁶ It relies on the notation of a *system of representatives*, where, for a finite universe \mathcal{U} and a set system $S \subseteq 2^{\mathcal{U}}$, a *system of representatives of S* is any set $\{u_H\}_{H \in S}$ with $u_H \in H$ for $H \in S$. In words, a system of representatives is obtained by replacing each set in S by an element in that set (its representative). (Note that an element can be chosen more than once as a representative, but, as defined above, only appears once in the system of representatives.)

► **Definition 10** (Rado matroid). *Let \mathcal{U} be a finite universe, $\mathcal{H} \subseteq 2^{\mathcal{U}}$ be some set system, and let $M = (\mathcal{H}, \mathcal{I})$ be a matroid. The Rado matroid $(\mathcal{U}, \bar{\mathcal{I}})$ induced by $(\mathcal{U}, \mathcal{H}, M)$ is a matroid on the ground set \mathcal{U} with independent sets*

$$\{U \subseteq \mathcal{U} : U \text{ is a system of representatives for some } I \in \mathcal{I}\} .$$

A proof that a Rado matroid is indeed a matroid can be found, e.g., in [13, Section 8.2]. We will reduce \mathcal{F} -CP to XWI on a Rado matroid obtained from a linear matroid. For this, we need that also the Rado matroid we obtain is linear and, moreover, that an explicit linear representation of it can be found efficiently, which is the case due to a result from [12].

► **Lemma 11** (see Theorem 3 of [12]). *For a set family $\mathcal{H} \subseteq 2^{\mathcal{U}}$ and a linear matroid $M = (\mathcal{H}, \mathcal{I})$, the Rado matroid $\bar{M} = (\mathcal{U}, \bar{\mathcal{I}})$ induced by $(\mathcal{U}, \mathcal{H}, M)$ is a linear matroid. Moreover, given a linear representation of M , one can find a linear representation of \bar{M} in time polynomial in $|\mathcal{H}|$, $|\mathcal{U}|$, and the size of the linear representation of M .*

We are now ready to show that \mathcal{F} -CP can be solved efficiently for linear matroids, which implies Theorem 4.

► **Lemma 12.** *\mathcal{F} -CP can be solved efficiently when \mathcal{F} is the family of independent sets of a linear matroid.*

Proof. We recall that we are given an \mathcal{F} -CP instance, which defines a set system $\mathcal{H} \subseteq 2^{\mathcal{U}}$ over a finite universe \mathcal{U} , and a family $\mathcal{F} \subseteq 2^{\mathcal{H}}$ such that $M = (\mathcal{H}, \mathcal{F})$ is a linear matroid. Let $\bar{M} = (\mathcal{U}, \bar{\mathcal{I}})$ be the Rado matroid induced by $(\mathcal{U}, \mathcal{H}, M)$. \bar{M} is a linear matroid by Lemma 11 and we can obtain a linear representation of \bar{M} in polynomial time. The promise of \mathcal{F} -CP implies the existence of an independent set T of \bar{M} satisfying the covering requirements, i.e.,

$$w_\ell(T) \geq m_\ell \quad \forall \ell \in [\gamma] . \tag{1}$$

To solve \mathcal{F} -CP, we guess, for each color $\ell \in [\gamma]$, the weight $\lambda_\ell := w_\ell(T)$ that T covers. Note that λ_ℓ is at most $W_\ell := w_\ell(\mathcal{U})$, which, due to the unary encoding of w_ℓ , is polynomially bounded in the input. Hence, the guessing of the λ_ℓ , for $\ell \in [\gamma]$, can be performed in time $\prod_{\ell \in [\gamma]} W_\ell$, which is polynomially bounded because $\gamma = O(1)$.

We now determine an independent set \tilde{T} in \bar{M} with $w_\ell(\tilde{T}) = \lambda_\ell$ for each $\ell \in [\gamma]$. This can be achieved by encoding all ℓ many (unary encoded) weight functions w_ℓ for $\ell \in [\gamma]$ into a single one \bar{w} and then solving an appropriate XWI problem with respect to \bar{w} . More precisely,

⁶ This construction of Rado matroids is also called the *induction of a matroid by a bipartite graph*.

for an element $u \in \mathcal{U}$, we obtain a new single weight $\bar{w}(u)$ whose first $\lceil \log_2(|W_1| + 1) \rceil$ bits represent the weight $w_1(u)$, the next $\lceil \log_2(|W_2| + 1) \rceil$ bits the weight $w_2(u)$, and so on. Because $\gamma = O(1)$ and all w_ℓ have unary encoding, this leads to combined weights \bar{w} whose unary encoding is polynomially bounded. Analogously, we encode the guessed weights λ_ℓ for $\ell \in [\gamma]$ into a single one $\bar{\lambda}$. We now solve XWI on \bar{M} with weights \bar{w} and target weight $\bar{\lambda}$. As \bar{M} is linear, this is possible by a randomized algorithm in time pseudo-polynomial in the total weight [4]. Moreover, because the weights are unary encoded in our setting, this implies a polynomial running time as desired.

Let \tilde{T} be a solution of this XWI problem, which must exist for the correct guess of the λ_ℓ because of the promised solution T . \tilde{T} being independent in \bar{M} implies that it is a system of representatives for some independent set $S \in \mathcal{F}$ of M . Such a set S can be found through matroid intersection. More precisely, it is known that the minimal (inclusion-wise) sets $I \subseteq \mathcal{H}$ such that \tilde{T} is a system of representatives for I form the basis of a matroid \tilde{M} , for which an efficient independence oracle can be obtained. (See [13, Section 7.3].) Hence, the desired set S can be obtained by finding a basis of \tilde{M} that is independent in M , which can be computed through matroid intersection algorithms. The set S is the solution of \mathcal{F} -CP that we return. Because $\tilde{T} \subseteq \bigcup_{H \in S} H$, the set S fulfills the covering requirements due to (1). ◀

3.2 γ -Colorful Knapsack Supplier

To showcase the versatility of our reduction, we now show how it implies an $O(2^\gamma)$ -approximation for γ -Colorful Knapsack Supplier, by discussing an efficient way to solve \mathcal{F} -CP when \mathcal{F} are the feasible solutions to a knapsack constraint. Even though there is a stronger (and more sophisticated) approximation result for this problem (as stated in Theorem 6), this application is a nice example of how one can readily obtain constant-factor approximations through our reduction technique combined with known methods; in this case, by solving \mathcal{F} -CP through a standard dynamic programming approach.

► **Lemma 13.** *Let \mathcal{F} be the feasible sets of a knapsack constraint, i.e., $\mathcal{F} = \{S \subseteq \mathcal{H} : \kappa(S) \leq K\}$ for some $\kappa : \mathcal{H} \rightarrow \mathbb{R}_{\geq 0}$ and budget $K \in \mathbb{R}_{\geq 0}$. Then \mathcal{F} -CP can be solved efficiently.*

Proof. Recall that the \mathcal{F} -CP problem to be solved defines a family $\mathcal{H} \subseteq 2^{\mathcal{U}}$ over a finite universe \mathcal{U} , and a family $\mathcal{F} \subseteq 2^{\mathcal{H}}$, which is defined by a knapsack constraint, i.e., $\mathcal{F} = \{S \subseteq \mathcal{H} : \kappa(S) \leq K\}$. We define the following weight function on \mathcal{U} :

$$\eta(u) := \min\{\kappa(H) : H \in \mathcal{H} \text{ with } u \in H\} .$$

In words, $\eta(u)$ corresponds to the cost of the cheapest set in \mathcal{H} that covers u . Consider the following binary program, which can be solved efficiently by standard dynamic programming techniques due to the unary encoding of the weights w_ℓ for $\ell \in [\gamma]$ (see, e.g., [1] for details):

$$\begin{aligned} \min \quad & \sum_{u \in \mathcal{U}} \eta(u) \cdot z(u) \\ & \sum_{u \in \mathcal{U}} w_\ell(u) \cdot z(u) \geq m_\ell \quad \forall \ell \in [\gamma] \\ & z \in \{0, 1\}^{\mathcal{U}} . \end{aligned}$$

We compute an optimal solution z^* to the above binary program. Let $Q := \{u \in \mathcal{U} : z^*(u) = 1\}$. For each $u \in Q$, let $H_u \in \mathcal{H}$ be a set of minimum cost that contains u ; hence, $\kappa(H_u) = \eta(u)$. We claim that $\{H_u : u \in Q\}$ is a solution to \mathcal{F} -CP. Because

z^* fulfills the constraints of the binary program, we have that $\{H_u : u \in Q\}$ fulfills the covering requirements. It remains to show that it fulfills the knapsack constraint, i.e., its cost is at most K . This reduces to show that the optimal value of the binary program is at most K . We claim that this holds because of the promise of \mathcal{F} -CP. Indeed, the promise guarantees that there is $S \subseteq \mathcal{F}$ and a system of representatives u_H for $H \in S$ such that $w_\ell(\{u_H : H \in S\}) \geq m_\ell$ for $\ell \in [\gamma]$. Hence, setting $z_{u_H} = 1$ for all $H \in S$, and setting all other coordinates of $z \in \{0, 1\}^{\mathcal{U}}$ to zero, is a solution to the binary program which has objective value at most $\kappa(S) \leq K$. \blacktriangleleft

4 Existence and construction of strong (L, r) -partitions

We now prove our key structural result, Lemma 9, which guarantees the existence and efficient constructability of $(O(2^\gamma), r)$ -partitions for γ -colorful spaces. Our proof proceeds by induction on γ . The base case, i.e., $\gamma = 0$, holds because the family $\{\{c\} : c \in C\}$ is a $(0, r)$ -grouping on every 0-colorful space $(C \dot{\cup} F, d, w)$. The key step is extending an (L, r) -partition of a $(\gamma - 1)$ -colorful space to a suitable partition of a γ -colorful space.

To this end, we extend ideas on the greedy algorithm of [6], which was originally introduced to deal with a single color k -center problem. More precisely, to augment a partition of a $(\gamma - 1)$ -colorful space, we apply a greedy subroutine on the points of color γ . A careful construction and analysis (which takes into account the earlier colors) then shows that this yields a $(2L + 10, r)$ -partition of the γ -colorful space. Our refined charging scheme improves on a decoupled analysis of [10] (which gives an $O(5^\gamma)$ approximation algorithm for γ -Colorful k -Center).

The lemma below formalizes the induction step.

► **Lemma 14.** *Given a (L, r) -partition for a $(\gamma - 1)$ -colorful space, then one can efficiently construct a $(2L + 10, r)$ -partition for any γ -colorful space obtained by adding one color to the $(\gamma - 1)$ -colorful space.*

Proof. Let $(C \dot{\cup} F, d, w)$ be a γ -colorful space, and let $\widehat{w} = (w_1, \dots, w_{\gamma-1})$ be the first $\gamma - 1$ colors. (Hence, we omitted the last color.) Let $C_\gamma := \text{supp}(w_\gamma)$ and $C_{<\gamma} := C \setminus C_\gamma$, and let \mathcal{P} be a (L, r) -partition of the $(\gamma - 1)$ -colorful space $(C_{<\gamma} \dot{\cup} F, d, \widehat{w})$. Note that we assumed that the supports of the weights w_ℓ are disjoint. Hence, $w_\ell(C_\gamma) = 0$ for $\ell \in [\gamma - 1]$. Moreover, without loss of generality, we assume that for every client $c \in C$, there is a facility $f \in F$ with $d(f, c) \leq r$. All clients not fulfilling this condition can be deleted from the instance without changing the statement as they can never be covered by any radius- r solution. Indeed, a partition of the clients of this purged instance can simply be extended to a partition of all clients by adding the deleted clients as singleton sets to the partition.

We now prove that Algorithm 1 returns an (\overline{L}, r) -partition $\overline{\mathcal{P}}$ of $(C \dot{\cup} F, d, w)$, where $\overline{L} := 2L + 10$. Algorithm 1 goes through all facilities in a well-chosen order and iteratively builds new parts consisting of parts in \mathcal{P} together with a subset of C_γ . (See Figure 2 for an illustration of this procedure.)

First, observe that $\overline{\mathcal{P}}$ is a partition. It clearly covers all clients as no client is farther than distance r away from its nearest facility, and we consider all facilities. Moreover, the sets in $\overline{\mathcal{P}}$ are disjoint by construction. Now, observe that any $\overline{A}_i \in \overline{\mathcal{P}}$ has small diameter, because

$$\text{diam}(\overline{A}_i) \leq 2 \cdot \max_{c \in \overline{A}_i} d(g_i, c) \leq 10r + 2Lr \quad ,$$

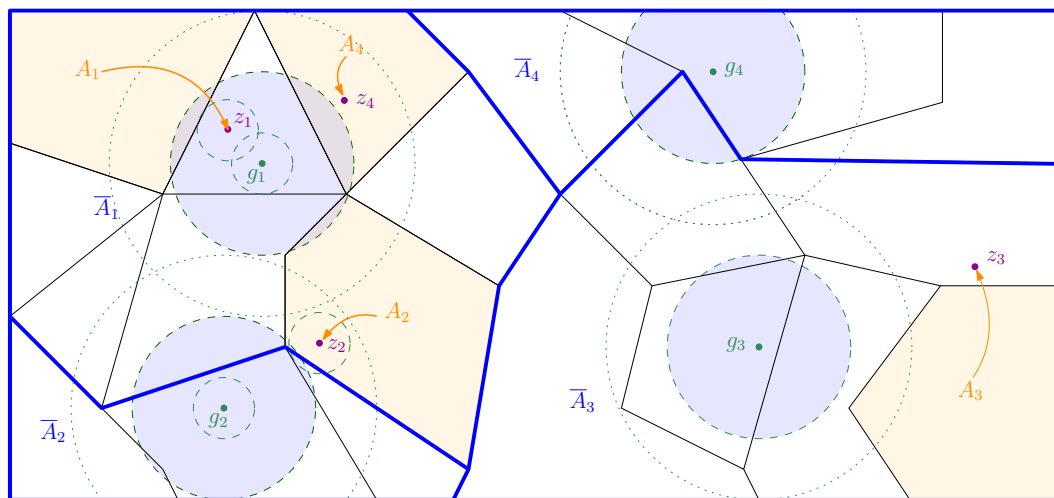
where the second inequality holds because $d(g_i, c) \leq 5r + Lr$ for any $c \in \overline{A}_i$ due to the following. Consider $c \in \overline{A}_i$. If $c \in C_\gamma$, then we even have $d(g_i, c) \leq 3r$. Otherwise, let

■ **Algorithm 1** GREEDYPARTITIONING($C, F, d, \mathcal{P}, w_\gamma$).

```

for  $i_{A_i} = 1$  to  $|F|$  do
   $g_i = \operatorname{argmax}_{f \in F \setminus \{g_1, \dots, g_{i-1}\}} w_\gamma \left( B_C(f, r) \setminus \bigcup_{t=1}^{i-1} \bar{A}_t \right);$ 
   $\bar{A}_i \leftarrow \left( B_{C_\gamma}(g_i, 3r) \cup \bigcup_{\substack{A \in \mathcal{P} \text{ with} \\ d(g_i, A) \leq 5r}} A \right) \setminus \bigcup_{t=1}^{i-1} \bar{A}_t;$ 
end
return  $\bar{\mathcal{P}} := \{\bar{A}_i : i \in [|F|]\};$ 

```



■ **Figure 2** Visualization of an (L, r) -partition and of Algorithm 2. The black polygons depict an (L, r) -partition \mathcal{P} of the clients $C_{<\gamma}$. The blue polygons shows how the clients $C_{<\gamma}$ are partitioned by $\bar{\mathcal{P}}$. Moreover, the blue $3r$ -balls around g_i illustrate which clients of C_γ get assigned to the part $\bar{A}_i \in \bar{\mathcal{P}}$. The dashed circles have radius r and $3r$ respectively, while the dotted circles have radius $5r$. We assume $Z = \{z_i \mid i \in [4]\}$ is given and we construct the respective $\bar{\mathcal{A}}$ and \bar{h} , given \mathcal{A} and h . We have $\mathcal{A} = \{A_i \mid i \in [4]\}$ (the orange areas) and $\bar{\mathcal{A}} = \{\bar{A}_i \mid i \in [3]\}$. Moreover $h(A_i) = z_i$ for $i \in [4]$ (depicted by an orange arrow), while $\bar{h}(\bar{A}_i) = z_i$ for $i \in [3]$.

$A \in \mathcal{P}$ be the set in the partition \mathcal{P} containing c . Note that $c \in \bar{A}_i$ implies $A \subseteq \bar{A}_i$. Hence, $d(g_i, c) \leq d(g_i, A) + \max\{d(b, c) : b \in A\} \leq 5r + Lr$, where we use $d(g_i, A) \leq 5r$, because $A \subseteq \bar{A}_i$, and $\text{diam}(A) \leq Lr$, which holds because \mathcal{P} is an (L, r) -partition. Thus, property 1 of the definition of an $(2L + 10, r)$ -partition (Definition 8) is fulfilled for $\bar{\mathcal{P}}$.

It remains to show that property 2 holds for a given selection Z . To this end, we use that \mathcal{P} is an (L, r) -partition, which implies that there is a subfamily $\mathcal{A} \subseteq \mathcal{P}$ and a corresponding injection $h : \mathcal{A} \rightarrow Z$ fulfilling property 2 of Definition 8 for the $(\gamma - 1)$ -colorful space $(C_{<\gamma} \cup F, d, \hat{w})$. In the following we construct $\bar{\mathcal{A}} \subseteq \bar{\mathcal{P}}$ and $\bar{h} : \bar{\mathcal{A}} \rightarrow Z$ such that property 2 of Definition 8 is satisfied for $\bar{\mathcal{A}}$ and \bar{h} . At the same time when constructing $\bar{\mathcal{A}}$, we employ a careful charging argument that makes sure that $w_\gamma(\bigcup_{\bar{A} \in \bar{\mathcal{A}}} \bar{A}) \geq w_\gamma(B_C(Z, r))$, i.e., that the constructed $\bar{\mathcal{A}}$ covers at least as much as Z of color γ . For the remaining colors, we show that the new selection $\bar{\mathcal{A}}$ includes all of \mathcal{A} ; formally, we show that for each $A \in \mathcal{A}$, there is an $\bar{A} \in \bar{\mathcal{A}}$ such that $A \subseteq \bar{A}$. This, as well as $d(\bar{A}, \bar{h}(\bar{A})) \leq r$ for all $\bar{A} \in \bar{\mathcal{P}}$ and injectivity of \bar{h} , are proved later.

7:12 Techniques for Generalized Colorful k -Center Problems

For $i \in [|F|]$, we define

$$U_i := C \setminus \bigcup_{t=1}^{i-1} \bar{A}_t$$

to be the clients that are “uncovered” at step i . By the way Algorithm 1 selects g_i in each iteration $i \in [|F|]$, we have

$$w_\gamma(B_C(g_i, r) \cap U_i) \geq w_\gamma(B_C(f, r) \cap U_i) \quad \forall i \in [|F|] \text{ and } f \in F,$$

which we call the *greediness property*.

We now describe the construction of $\bar{\mathcal{A}}$ and the charging scheme in detail. We successively add sets $\bar{A}_i \in \bar{\mathcal{P}}$ to $\bar{\mathcal{A}}$, where the sets \bar{A}_i are considered in increasing order of their index. When adding a set \bar{A}_i to $\bar{\mathcal{A}}$, we also perform two further steps: (i) we identify an element $f \in Z$ and set $\bar{h}(\bar{A}_i) = f$, and (ii) we mark f as assigned to make sure that we never assign it again in the future (as \bar{h} needs to be an injection). For convenience, for $i \in [|F|]$ and $f \in Z$, we write $\text{ASSIGN}(i, f)$ for performing these steps, i.e., adding \bar{A}_i to $\bar{\mathcal{A}}$, setting $\bar{h}(\bar{A}_i)$ to f , and marking f as assigned.

The charging argument charges the coverage of color γ of $B_C(Z, r)$ against the γ -coverage in $\bigcup_{\bar{A} \in \bar{\mathcal{A}}} \bar{A}$. Whenever we charge a set $Q \subseteq B_C(Z, r)$ against some subset $W \subseteq \bigcup_{\bar{A} \in \bar{\mathcal{A}}} \bar{A}$, we make sure that $w_\gamma(Q) \leq w_\gamma(W)$. Algorithm 2 shows our procedure to construct both $\bar{\mathcal{A}}$ and the desired injection $\bar{h}: \bar{\mathcal{A}} \rightarrow Z$ together with the charging argument. (See also Figure 2.)

■ **Algorithm 2** Construction of $\bar{\mathcal{A}}$ and injection $\bar{h}: \bar{\mathcal{A}} \rightarrow Z$ together with charging argument.

Mark all facilities in Z as unassigned.

for $i = 1$ **to** $|F|$ **do**

- **Rule 1** If there is an unassigned $f \in Z$ with $B_C(f, r) \cap B_C(g_i, r) \cap U_i \neq \emptyset$:
ASSIGN(i, f).
- **Rule 2** Else if there is an unassigned $f \in Z$ with $B_C(f, r) \cap B_C(g_i, 3r) \cap U_i \neq \emptyset$:
ASSIGN(i, f) and charge $B_C(f, r) \cap U_i$ against $B_C(g_i, r) \cap U_i$.
- **Rule 3** Else if there is an $A \in \mathcal{A}$ such that $A \subseteq \bar{A}_i$:
ASSIGN($i, h(A)$) and charge $B_C(h(A), r) \cap U_i$ against $B_C(g_i, r) \cap U_i$.

If ASSIGN was called, charge against themselves all points in \bar{A}_i that have not been charged yet.

end

We start by showing that \bar{h} is an injection. Suppose f is assigned using Rule 1 or 2. Then f was not assigned so far as we only assign unassigned facilities. Now suppose $h(A) = \bar{h}(\bar{A}_i)$ is assigned using Rule 3. We claim that $h(A)$ is not assigned so far. Assume by the sake of deriving a contradiction that it was assigned in a previous iteration $j < i$. It cannot have been assigned by Rule 3, since h is injective. So assume it was assigned by Rule 1 or 2. Hence, g_j satisfies $B_C(g_j, 3r) \cap B_C(h(A), r) \cap U_j \neq \emptyset$. This implies that $d(g_j, h(A)) \leq 4r$ and thus $A \subseteq \bar{A}_j$, which contradicts $A \subseteq \bar{A}_i$.

Moreover, \bar{A}_i fulfills property 2a of a $(2L + 10, r)$ -partition because of the following. Let $f \in Z$ and $\bar{A}_i := \bar{h}^{-1}(f)$, and we have to show that $d(\bar{A}_i, f) \leq r$. Because $h(\bar{A}_i) = f$, we called at some point during Algorithm 2 the procedure $\text{ASSIGN}(i, f)$. In both Rule 1 and Rule 2 we have $B_C(f, r) \cap B_C(g_i, 3r) \cap U_i \neq \emptyset$, which implies that \bar{A}_i contains a client in $B_C(f, r)$, as desired. If $\text{ASSIGN}(i, f)$ was called in Rule 3, then we have $h^{-1}(f) \subseteq \bar{A}_i$, which implies $d(\bar{A}_i, f) \leq d(h^{-1}(f), f) \leq r$ by the fact that \mathcal{P} is an (L, r) -partition.

It remains to show that \bar{A} fulfills property 2b of an $(2L+10, r)$ -partition. We first consider the last color (color γ) and show $w_\gamma(\bigcup_{\bar{A} \in \bar{\mathcal{A}}} \bar{A}) \geq w_\gamma(B_C(Z, r))$. To this end, observe that the charging indeed charges clients in $B_C(Z, r)$ against clients in $\bigcup_{\bar{A} \in \bar{\mathcal{A}}} \bar{A}$. We allow for charging a client in $B_C(Z, r)$ against more than one client in $\bigcup_{\bar{A} \in \bar{\mathcal{A}}} \bar{A}$. However, no client in $\bigcup_{\bar{A} \in \bar{\mathcal{A}}} \bar{A}$ gets charged against more than once because in iteration i we only charge against clients in \bar{A}_i , and the sets $\bar{\mathcal{A}} = \{\bar{A}_1, \dots, \bar{A}_{|F|}\}$ form a partition of C . Also note that we always charge clients of $B_C(Z, r)$ against clients of $\bigcup_{\bar{A} \in \bar{\mathcal{A}}} \bar{A}$ of at least the same w_γ -weight. This is true whenever charging happens in Rule 2 or Rule 3, because of the greediness property, and holds trivially for all other charging operations, which only charge clients against themselves. To conclude that $w_\gamma(\bigcup_{\bar{A} \in \bar{\mathcal{A}}} \bar{A}) \geq w_\gamma(B_C(Z, r))$, it remains to observe that all of $B_C(Z, r)$ gets charged against something.

To this end, fix a facility $f \in Z$. Consider an iteration j of Algorithm 2 such that $B_C(g_j, 3r) \cap U_j$ intersects $B_C(f, r)$. We claim that for each such iteration, either $\text{ASSIGN}(j, f)$ is called, or $B_C(f, r) \cap B_C(g_j, 3r) \cap U_j$ is charged. To prove the claim, suppose f is not assigned in iteration j . By Algorithm 2, either Rule 1 or Rule 2 must have applied in this iteration j , as f satisfies the condition of Rule 2. Thus ASSIGN was called on j and all points in \bar{A}_j have been charged. Now suppose the first case applies, i.e., $\text{ASSIGN}(j, f)$ is called for some j . Then all of $B_C(f, r) \cap U_j$ is charged (and $B_C(f, r) \setminus U_j$ is already charged by the second case). If the first case never applies, then all of $B_C(f, r)$ is charged by the second case since $U_{|F|}$ is empty. Hence, all of $B_C(f, r)$ is charged, as desired.

To see that property 2b of Definition 8 is fulfilled also for all colors $\ell \in [\gamma - 1]$, observe that Rule 3 makes sure that any component that was in \mathcal{A} will still be selected in $\bar{\mathcal{A}}$. Thus, $w_\ell(\bar{\mathcal{A}}) \geq w_\ell(B_C(Z, r))$ for all colors $\ell \in [\gamma]$.

It remains to show that $d(\bar{A}_i, \bar{h}(\bar{A}_i)) \leq r$. If Rule 1 or Rule 2 is applied, this is satisfied as there is a client $c \in B_C(g_i, 3r) \cap B_C(\bar{h}(\bar{A}_i), r) \cap U_i$; because $c \in \bar{A}_i$ by construction, we have $d(\bar{h}(\bar{A}_i), \bar{A}_i) \leq d(\bar{h}(\bar{A}_i), c) \leq r$. If Rule 3 is applied for $A \subseteq \bar{A}_i$, we also have $d(\bar{h}(\bar{A}_i), \bar{A}_i) \leq d(\bar{h}(\bar{A}_i), A) = d(h(A), A) \leq r$, where the last inequality follows from \mathcal{P} being an (L, r) -partition. \blacktriangleleft

Lemma 9 now follows readily from Lemma 14.

Proof of Lemma 9. The proof follows by induction on γ . For the induction start, consider $\gamma = 0$. The set $\{\{c\} : c \in C\}$ is a $(0, r)$ -partition on every 0-colorful space $(C \cup F, d, w)$. The induction step is given by Lemma 14. Note that $2(10(2^{\gamma-1} - 1)) + 10 = 10(2^\gamma - 1)$. The running time is clearly $O(\text{poly}(|X|, \gamma))$ as every step in the induction takes time $O(\text{poly}(|X|, \gamma))$.⁷ \blacktriangleleft

References

- 1 G. Anegg, H. Angelidakis, A. Kurpisz, and R. Zenklusen. A technique for obtaining true approximations for k-center with covering constraints. *Mathematical Programming*, 2021. doi:10.1007/s10107-021-01645-y.
- 2 T. Bajpai, D. Chakrabarty, C. Chekuri, and M. Negahbani. Revisiting priority k-center: Fairness and outliers. In N. Bansal, E. Merelli, and J. 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 21:1–21:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.21.

⁷ As briefly mentioned earlier, one can obtain slightly better constants, leading to existence and conductivity of $(8 \cdot 2^\gamma - 10, r)$ -partitions. This can be achieved by using $\gamma = 1$ as base case, for which our techniques can be shown to imply that there are $(6, r)$ -partitions. In the interest of simplicity, we use the slightly weaker bound in Lemma 9.

- 3 S. Bandyopadhyay, T. Inamdar, S. Pai, and K. R. Varadarajan. A constant approximation for colorful k -center. In M. A. Bender, O. Svensson, and G. Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPICs*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ESA.2019.12.
- 4 P.M. Camerini, G. Galbiati, and F. Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992. doi:10.1016/0196-6774(92)90018-8.
- 5 D. Chakrabarty and M. Negahbani. Generalized center problems with outliers. *ACM Trans. Algorithms*, 15(3):41:1–41:14, 2019. doi:10.1145/3338513.
- 6 M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In S. Rao Kosaraju, editor, *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*, pages 642–651. ACM/SIAM, 2001. URL: <http://dl.acm.org/citation.cfm?id=365411.365555>.
- 7 D. Z. Chen, J. Li, H. Liang, and H. Wang. Matroid and knapsack center problems. *Algorithmica*, 75(1):27–52, 2016. doi:10.1007/s00453-015-0010-1.
- 8 F. Chierichetti, R. Kumar, S. Lattanzi, and S. Vassilvitskii. Fair clustering through fairlets. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5029–5037, 2017. URL: <https://proceedings.neurips.cc/paper/2017/hash/978f5b5bcc4eccc88ad48ce3914124a2-Abstract.html>.
- 9 D. G. Harris, T. W. Pensyl, A. Srinivasan, and K. Trinh. A lottery model for center-type problems with outliers. *ACM Trans. Algorithms*, 15(3):36:1–36:25, 2019. doi:10.1145/3311953.
- 10 T. Inamdar and K. Varadarajan. Non-uniform k -center and greedy clustering, 2021. arXiv: 2111.06362.
- 11 X. Jia, K. Sheth, and O. Svensson. Fair colorful k -center clustering. *Mathematical Programming*, 2021. doi:10.1007/s10107-021-01674-7.
- 12 M. J. Piff and D. J. A. Welsh. On the vector representation of matroids. *Journal of The London Mathematical Society-second Series*, pages 284–288, 1970.
- 13 D.J.A. Welsh. *Matroid theory*. Courier Corporation, 2010.

Simple Streaming Algorithms for Edge Coloring

Mohammad Ansari ✉

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Mohammad Saneian ✉

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Hamid Zarrabi-Zadeh ✉

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Abstract

We revisit the classical edge coloring problem for general graphs in the streaming model. In this model, the input graph is presented as a stream of edges, and the algorithm must report colors assigned to the edges in a streaming fashion, using a memory of size $O(n \text{ polylog } n)$ on graphs of up to $O(n^2)$ edges. In ESA 2019 and SOSA 2021, two elegant randomized algorithms were presented for this problem in the adversarial edge arrival model, where the latest one colors any input graph using $O(\Delta^2/s)$ colors with high probability in $\tilde{O}(ns)$ space. In this short note, we propose two extremely simple streaming algorithms that achieve the same color and space bounds deterministically. Besides being surprisingly simple, our algorithms do not use randomness at all, and are very simple to analyze.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Graph coloring

Keywords and phrases Edge coloring, streaming model, adversarial order

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.8

1 Introduction

An edge coloring of a graph G is an assignment of colors to the edges such that no two adjacent edges receive the same color. The chromatic index of a graph G is the smallest number of colors required to properly color edges of G . Although determining the chromatic index is NP-hard [10], a classic theorem by Vizing states that any graph can be colored with at most $\Delta + 1$ colors, where Δ is the maximum degree of the graph. On the other hand, Δ colors are always required to properly color edges of a graph.

In this short note, we revisit the edge coloring problem in the streaming model, where edges of the input graph are presented to the algorithm one at a time in an adversarially chosen order. In this model, the memory available to the algorithm is less than the input size, and hence, we cannot store all the edges of the graph. We consider one-pass algorithms in which the amount of available space is $O(n \text{ polylog } n)$, which is also known as the semi-streaming model [9]. Moreover, since output in the edge coloring problem is as large as the input, we cannot wait to report the output after the whole stream is processed. Instead, we need to report colors of the edges in a streaming fashion, a typical approach usually referred to as W-streaming in the literature [8].

The edge coloring problem has been studied in various models of computation. In the offline model, there are polynomial-time algorithms that compute $(\Delta + 1)$ -edge colorings for general graphs [1, 11]. In the online model, Bar-Noy et al. [3] were the first to show that no algorithm can do better than the greedy algorithm, which uses at most $2\Delta - 1$ colors, no matter if the input graph is revealed edge by edge or vertex by vertex. However, their lower bound only holds when $\Delta = O(\log n)$. Therefore, research has been shifted towards online edge coloring of higher-degree graphs, i.e., when $\Delta = \omega(\log n)$. In particular, Cohen et al. [7] achieved the first positive result by giving an algorithm that uses $(1 + o(1))\Delta$ colors for



© Mohammad Ansari, Mohammad Saneian, and Hamid Zarrabi-Zadeh;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 8; pp. 8:1–8:4



Leibniz International Proceedings in Informatics

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

■ **Table 1** Summary of streaming algorithms for edge coloring in the adversarial edge arrival model. Here, $s = o(\Delta)$ is a space parameter.

ALGORITHM	TYPE	# COLORS	SPACE	REF
Behnezhad et al.	randomized	$O(\Delta^2)$ (w.h.p.)	$\tilde{O}(n)$	[4]
Charikar and Liu	randomized	$(1 + o(1))\Delta^2/s$ (w.h.p.)	$\tilde{O}(ns)$	[6]
This work	deterministic	$(1 + o(1))\Delta^2/s$	$O(ns)$	[here]

bipartite graphs in the adversarial vertex arrival setting. In the same setting, Saberi and Wajc [12] have recently presented a $(1.9 + o(1))\Delta$ algorithm for general graphs. In the random-order edge arrival, where edges of the graph are presented in a uniformly random order, the best current algorithm uses $(1 + o(1))\Delta$ colors due to Bhattacharia et al. [5], improving upon the previous 1.26Δ algorithm of Bahmani et al. [2].

In the streaming model, existing results in the online model cannot be typically applied due to memory restriction. Behnezhad et al. [4] were the first to provide $O(n \text{ polylog } n)$ -space streaming algorithms for the edge coloring problem. They proposed an algorithm reporting a $2e\Delta$ -coloring in the random edge arrival case, and another $O(\Delta^2)$ algorithm, when edges arrive in an arbitrary (adversarial) order. Charikar and Liu [6] improved these results by presenting an algorithm that uses $(1 + o(1))\Delta$ colors on random streams, and another algorithm in the adversarial edge arrival case that uses $O(\Delta^2/s)$ colors with high probability in $\tilde{O}(ns)$ space.

In this work, we focus on the adversarial edge arrival model, where edges of the input graph are revealed in an arbitrary order. In this setting, the algorithm of Behnezhad et al. [4] randomly decomposes the input graph into $O(\log n)$ bipartite subgraphs, and uses a distinct palette of colors for each subgraph. In particular, for each vertex v and each bipartite subgraph i , a counter $C_{v,i}$ is stored to denote the degree of v in the subgraph i . For each incoming edge (u, v) belonging to subgraph i , the color $(C_{u,i}, C_{v,i}, i)$ is assigned to that edge, and both $C_{u,i}$ and $C_{v,i}$ are incremented by one. Behnezhad et al. showed that with high probability, the number of colors used this way is $O(\Delta^2)$. Charikar and Liu [6] used a similar technique, but modified the way each edge is randomly assigned to bipartite subgraphs. They showed that their new algorithm uses $O(\Delta^2/s)$ colors with high probability, when available space is $\tilde{O}(ns)$.

In this short note, we propose two simple deterministic algorithms for the edge coloring problem in the adversarial arrival streams. Besides being extremely simple, our algorithms achieve the current state-of-the-art bounds on the space and number of colors, without using randomization. More precisely, our algorithms use $(1 + o(1))\Delta^2/s$ colors in the worst case using $O(ns)$ space, where $s = o(\Delta)$ ¹. To our knowledge, these are the first deterministic algorithms for the problem in the adversarial edge arrival streams. A summary of the results for edge coloring in the adversarial edge arrival model is presented in Table 1.

2 A Simple Greedy Algorithm

Our first algorithm uses a simple greedy approach to color edges of the input graph in an online manner. The algorithm simply colors each edge arrived with the first available color, and continues coloring until the memory is full. At that point, the algorithm deletes the most

¹ Note that when $s = \Omega(\Delta)$, an $O(ns)$ space is enough to store the entire graph.

frequently used color and discards all edges which use that color to free up some memory. The pseudocode of the algorithm is presented in Algorithm 1. Throughout the algorithm, we say that a color c is *available* for coloring an edge e , if no other edge incident to either endpoint of e has already received color c . Note that during the course of the algorithm, a deleted color will never be used again.

■ **Algorithm 1** A Simple Greedy Algorithm.

```

start with a color palette of size 1
for each edge  $e$  in the stream do
    if no color is available in the palette to color  $e$  then
        | add a new color to the palette
    end
    color  $e$  with an available color in the palette
    if the memory capacity is hit then
        | delete the most frequently used color  $c$  from the palette
        | remove all edges colored with  $c$  from memory
    end
end

```

► **Theorem 1.** *Algorithm 1 colors any input graph with $(1 + o(1))\Delta^2/s$ colors in $O(ns)$ space.*

Proof. It is easy to see that the algorithm reports a proper coloring. Consider two edges e_1 and e_2 incident to a vertex v , where e_1 arrives earlier than e_2 in the stream. Upon arrival of e_2 , if e_1 is still in memory, then its color will not be considered for coloring e_2 by the algorithm. On the other hand, if e_1 is not in memory, then its color has been permanently deleted from the palette, and hence will not be considered for coloring e_2 again.

To bound the number of colors used by the algorithm, first note that at any point of time during the execution of the algorithm, the color palette has size at most $2\Delta - 1$. This is because for any edge $e = (u, v)$ in the stream, at most $2(\Delta - 1) = 2\Delta - 2$ distinct colors are used by the edges incident to either u or v , as both endpoints have degree at most Δ . As such, a color palette of size $2\Delta - 1$ has at least one color available for coloring e , and hence, no new color is added to such palette by the algorithm. Therefore, for a memory of size ns , whenever the memory capacity is hit, the most popular color deleted by the algorithm has frequency at least $ns/2\Delta$. Thus, the total number of colors deleted during the course of the algorithm is at most $|E|/(ns/2\Delta)$, which is upper bounded by Δ^2/s , since $|E| \leq n\Delta/2$. Therefore, the total number of colors used by the algorithm is at most $(2\Delta - 1) + \Delta^2/s$ which is $(1 + o(1))\Delta^2/s$, for $s = o(\Delta)$. ◀

3 A Simple Buffering Algorithm

Algorithm 1 colors each arrived edge instantly in an online manner. In the streaming model, however, we can use our limited space to “buffer” a chunk of input stream before processing it and producing output. Using this buffering technique, we can achieve an even simpler algorithm. The idea behind our second algorithm is to simply buffer every chunk of $O(ns)$ edges and color it using a distinct palette of $\Delta + 1$ colors. We can use any offline $\Delta + 1$ coloring algorithm to color each chunk of the input. The pseudocode of our algorithm is presented in Algorithm 2.

► **Theorem 2.** *Algorithm 2 colors any input graph with $(1 + o(1))\Delta^2/s$ colors in $O(ns)$ space.*

8:4 Simple Streaming Algorithms for Edge Coloring

■ **Algorithm 2** A Simple Buffering Algorithm.

```
partition the stream into chunks of size  $ns$ 
for each chunk of edges do
  | color the chunk using a distinct palette of  $\Delta + 1$  colors
end
```

Proof. As a new palette is used for coloring each chunk, it is clear that the algorithm reports a proper edge coloring. To analyze the number of colors, we observe that the number of chunks is $|E|/ns$. Therefore, the total number of colors used is

$$\frac{|E|}{ns}(\Delta + 1) \leq \frac{n\Delta}{2ns}(\Delta + 1) = \left(\frac{1}{2} + o(1)\right)\Delta^2/s,$$

where the above inequality holds because $|E| \leq n\Delta/2$. ◀

► **Remark.** Algorithm 2 can be easily modified to work online by just replacing the offline $\Delta + 1$ coloring algorithm with an online $2\Delta - 1$ greedy one. The resulting algorithm uses $(1 + o(1))\Delta^2/s$ colors in the worst case.

References

- 1 Eshrat Arjomandi. An efficient algorithm for colouring the edges of a graph with $\Delta + 1$ colours. *Information Systems and Operational Research*, 20(2):82–101, 1982.
- 2 Bahman Bahmani, Aranyak Mehta, and Rajeev Motwani. Online graph edge-coloring in the random-order arrival model. *Theory of Computing*, 8(1):567–595, 2012.
- 3 Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. The greedy algorithm is optimal for on-line edge coloring. *Information Processing Letters*, 44(5):251–253, 1992.
- 4 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. Streaming and massively parallel algorithms for edge coloring. In *Proceedings of the 27th European Symposium on Algorithms*, volume 144 of *LIPICs*, pages 15:1–15:14, 2019.
- 5 Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms*, pages 2830–2842, 2021.
- 6 Moses Charikar and Paul Liu. Improved algorithms for edge colouring in the W-streaming model. In *Proceedings of the 4th SIAM Symposium on Simplicity in Algorithms*, pages 181–183, 2021.
- 7 Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science*, pages 1–25, 2019.
- 8 Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 714–723, 2006.
- 9 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *Proceedings of the 31st International Colloquium on Automata, Languages, and Programming*, volume 3142 of *LNCS*, pages 531–543, 2004.
- 10 Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.
- 11 Jayadev Misra and David Gries. A constructive proof of Vizing’s theorem. *Information Processing Letters*, 41(3):131–133, 1992.
- 12 Amin Saberi and David Wajc. The greedy algorithm is not optimal for on-line edge coloring. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming*, volume 198 of *LIPICs*, pages 109:1–109:18, 2021.

Computing Smallest Convex Intersecting Polygons

Antonios Antoniadis ✉

Department for Applied Mathematics, University of Twente, Enschede, The Netherlands

Mark de Berg ✉

Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Sándor Kisfaludi-Bak ✉ 

Department of Computer Science, Aalto University, Espoo, Finland

Antonis Skarlatos ✉ 

Department of Computer Science, Universiät Salzburg, Austria

Abstract

A polygon C is an *intersecting polygon* for a set \mathcal{O} of objects in \mathbb{R}^2 if C intersects each object in \mathcal{O} , where the polygon includes its interior. We study the problem of computing the minimum-perimeter intersecting polygon and the minimum-area convex intersecting polygon for a given set \mathcal{O} of objects. We present an FPTAS for both problems for the case where \mathcal{O} is a set of possibly intersecting convex polygons in the plane of total complexity n .

Furthermore, we present an exact polynomial-time algorithm for the minimum-perimeter intersecting polygon for the case where \mathcal{O} is a set of n possibly intersecting segments in the plane. So far, polynomial-time exact algorithms were only known for the minimum perimeter intersecting polygon of lines or of disjoint segments.

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

Keywords and phrases convex hull, imprecise points, computational geometry

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.9

Related Version *Full Version*: <https://arxiv.org/abs/2208.07567>

Funding Mark de Berg is supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.

Antonis Skarlatos: Part of the work was done during an internship at the Max Planck Institute for Informatics in Saarbrücken, Germany.

1 Introduction

Convex hulls are among the most fundamental objects studied in computational geometry. In fact, the problem of designing efficient algorithms to compute the convex hull of a planar point set \mathcal{O} – the smallest convex set containing \mathcal{O} – is one of the problems that started the field [5, 11]. Since the early days, the problem has been studied extensively, resulting in practical and provably efficient algorithms, in the plane as well as in higher dimensions; see the survey by Seidel [13, Chapter 26] for an overview.

A natural generalization is to consider convex hulls for a collection \mathcal{O} of geometric objects (instead of points) in \mathbb{R}^2 . Note that the convex hull of a set of polygonal objects is the same as the convex hull of the vertices of the objects. Hence, such convex hulls can be computed using algorithms for computing the convex hull of a point set. A different generalization, which leads to more challenging algorithmic questions, is to consider the smallest convex set that *intersects* all objects in \mathcal{O} . Thus, instead of requiring the convex set to fully contain each object from \mathcal{O} , we only require that it has a non-empty intersection with each object.

Notice that in case of points, the “smallest” set is well-defined: if convex sets C_1 and C_2 both contain a point set \mathcal{O} , then $C_1 \cap C_2$ also contains \mathcal{O} . Hence, the convex hull of a point set \mathcal{O} can be defined as the intersection of all convex sets containing \mathcal{O} . When \mathcal{O} consists of objects, however, this is no longer true, and the term “smallest” is ambiguous. In the present



© Antonios Antoniadis, Mark de Berg, Sándor Kisfaludi-Bak, and Antonis Skarlatos; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 9; pp. 9:1–9:13

Leibniz International Proceedings in Informatics



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

paper we consider two variants: given a set \mathcal{O} of possibly intersecting convex polygons in \mathbb{R}^2 of total complexity n , find a convex set of minimum perimeter that intersects all objects in \mathcal{O} , or a convex set of minimum area that intersects all objects in \mathcal{O} .

Observe that a minimum-perimeter connected intersecting set C for \mathcal{O} must be a convex polygon. To see this, observe that for any object $o \in \mathcal{O}$ we can select a point $p_o \in o \cap C$, and take the convex hull of these points; the result is a feasible convex polygon whose perimeter is no longer than that of C . Thus the convexity of the solution could be omitted from the problem statement. This contrasts with the minimum-area problem, where there is always an intersecting polygon of zero area, namely, a tree. The convexity requirement is therefore essential in the problem statement. Note that it is still true that the minimum-area convex intersecting set is a polygon: given a convex solution, we can again take the convex hull of the points p_o and get a feasible solution whose area is not greater than the area of the initial convex solution. We also remark that the two problems typically have different optima. If \mathcal{O} consists of the three edges of an equilateral triangle, then the minimum-area solution is a line segment (that is, a degenerate polygon of zero area), whereas the minimum-perimeter solution is the triangle whose vertices are the midpoints of the edges.

The problem of computing minimum-area or minimum-perimeter convex intersecting polygons, as well as several related problems, have already been studied. Dumitrescu and Jiang [4] considered the minimum-perimeter intersecting polygon problem. They gave a constant-factor approximation algorithm as well as a PTAS for the case when the objects in \mathcal{O} are segments or convex polygons. They achieved a running time of $n^{O(1)/\varepsilon} + 2^{O(1/\varepsilon^{2/3})}n$. They also prove that computing a minimum-perimeter intersecting polygon for a set \mathcal{O} of non-convex polygons (or polygonal chains) is NP-hard. For convex input objects, however, the hardness proof fails. Hence, Dumitrescu and Jiang ask the following question.

Question 1. Is the problem of computing a minimum-perimeter intersecting polygon of a set of segments NP-hard?

In case of *disjoint* segments, a minimum-perimeter intersecting polygon can be found in polynomial time [6, 7], but for intersecting segments the question is still open.

The problem of computing smallest intersecting polygons for a set \mathcal{O} of objects has also been studied in works on *imprecise points*. Now the input is a set of points, but the exact locations of the points are unknown. Instead, for each point one is given a region where the point can lie. One can then ask questions such as: what is the largest possible convex hull of the imprecise points? And what is the smallest possible convex hull? If we consider the objects in our input set \mathcal{O} as the regions for the imprecise points, then the latter question is exactly the same as our problem of finding smallest intersecting convex sets. In this setup both the minimum-perimeter and minimum-area problem have been considered, for sets \mathcal{O} consisting of convex regions of total complexity n . There are exact polynomial-time algorithms for minimum (and maximum) perimeter and area, for the special case where \mathcal{O} consists of horizontal line segments or axis-parallel squares [10]. Surprisingly, some of these problems are NP-hard, such as the *maximum*-area/perimeter problems for segments. This gave rise to the study of approximation algorithms and approximation schemes [9].

In some cases, the minimum-perimeter problem can be phrased as a travelling salesman problem with neighborhoods (TSPN). Here the goal is to find the shortest closed curve intersecting all objects from the given set \mathcal{O} . In general, an optimal TSPN tour need not be convex, but one can show that in the case of lines or rays, an optimal tour is always convex: if a convex polygon intersects a line (or a ray) then its boundary intersects the line (resp. the ray). Therefore, computing a minimum-perimeter intersecting polygon of lines (or rays) is the same problem as TSPN with line neighborhoods (resp. ray neighborhoods). TSPN of

lines in \mathbb{R}^2 admits a polynomial-time algorithm [2]. In higher dimensions, TSPN has a PTAS for hyperplane neighborhoods [1], but notice that this is not the natural generalization of the minimum-intersecting polygon problem. Tan [12] proposed an exact algorithm for TSPN of rays in \mathbb{R}^2 , but there seems to be an error in the argument; see the full version for details. At the time of writing this article, we believe that a polynomial-time algorithm for TSPN of rays is not known, but there is a constant-factor approximation algorithm due to Dumitrescu [3], as well as a PTAS [4].

Our results. In order to resolve Question 1, we first need to establish a good structural understanding and a dynamic programming algorithm. It turns out that the algorithm can also be used for approximation. We give dynamic-programming-based approximation schemes for the minimum-perimeter and minimum-area convex intersecting polygon problems. Our first algorithm is a fully polynomial time approximation scheme (FPTAS) for the minimum-perimeter problem of arbitrary convex objects of total complexity n .

► **Theorem 1.** *Let \mathcal{O} be a set of convex polygons of total complexity n in \mathbb{R}^2 and let OPT be the minimum perimeter of an intersecting convex polygon for \mathcal{O} . For any given $\varepsilon > 0$, we can compute an intersecting polygon for \mathcal{O} whose perimeter is at most $(1 + \varepsilon) \cdot \text{OPT}$, in $O(n^{2.373}/\varepsilon + n/\varepsilon^8)$ time.*

This is a vast improvement over the PTAS given by Dumitrescu and Jiang [4], as the dependence on $1/\varepsilon$ is only polynomial in our algorithm. Our approximation algorithms work in a word-RAM model, where input polygons are defined by the coordinates of their vertices, and where each coordinate is a word of $O(\log n)$ bits.

We also get a similar approximation scheme for the minimum area problem, albeit with a slower running time. Here we rely more strongly on the fact that the objects of \mathcal{O} are convex *polygons*, and an extension to (for example) disks is an interesting open question. The full version details how the minimum-perimeter FPTAS needs to be adapted to the minimum-area setting.

► **Theorem 2.** *Let \mathcal{O} be a set of convex polygons of total complexity n in \mathbb{R}^2 and let OPT be the minimum area of an intersecting convex polygon for \mathcal{O} . For any given $\varepsilon > 0$, we can compute a convex intersecting polygon for \mathcal{O} whose area is at most $(1 + \varepsilon) \cdot \text{OPT}$, in $O(n^{17} \log(1/\varepsilon) + n^{11}/\varepsilon^{24})$ time.*

We remark that both Theorem 1 and Theorem 2 work if the input has *polytopes* instead of polygons, that is, when each object is the intersection of some half-planes.

While the dynamic programming algorithm developed above is crucial to get an exact algorithm, we are still several steps from being able to resolve Question 1. The main challenge here is that the vertices of the optimum intersecting polygon can be located at arbitrary boundary points in \mathcal{O} , and there is no known way to discretize the problem. We introduce a subroutine that uses an algorithm of Dror et al. [2] to compute parts of the minimum-perimeter intersecting polygon that contain no vertices of input objects. We are able to achieve a polynomial-time algorithm (on a real-RAM machine) for the minimum perimeter intersecting polygon problem only when the objects are line segments.

► **Theorem 3.** *Let \mathcal{O} be a set of n line segments in the plane. Then we can compute a minimum-perimeter intersecting polygon for \mathcal{O} in $O(n^9 \log n)$ time.*

If $P \neq NP$, then this gives a direct negative answer to Question 1. The theorem also extends to the case of rays (this is the scenario studied by Tan [12]; see the discussion in the full version).

Our techniques. Our approximation algorithms both compute an approximate solution whose vertices are from some fine grid. To determine a suitable grid resolution, we need to be able to compute lower bounds on OPT , which is non-trivial. It is also non-trivial to know where to place the grid, such that it is guaranteed to contain an approximate solution. The problem is that our lower bound gives us the location of a solution that is a constant-factor approximation, but this can be far from the location of a $(1 + \varepsilon)$ -approximation. Hence, for the minimum-area problem we generate a collection of grids, one of which is guaranteed to contain a $(1 + \varepsilon)$ -approximate solution. Finally, we face some further difficulties since a square grid may be insufficient: the optimum intersecting polygon may be extremely (exponentially) thin and long, and of area close to zero. In such cases there is no square grid of polynomial size that would contain a good solution. These problems are resolved in Section 2.

Section 3 presents our dynamic programming algorithm for minimum perimeter. In the dynamic programming the main technical difficulty lies in the fact that it is not clear what subset of objects should be visited in each subproblem. A portion of the optimum’s boundary could in principle be tasked with intersecting an arbitrary subset of \mathcal{O} , while some of the objects in \mathcal{O} need not be intersected by the optimum boundary and will simply be covered by the interior of the optimum intersecting polygon: a naïve approach therefore would not yield a polynomial-time algorithm. Our carefully designed subproblems have a clear corresponding set of objects to “visit”, using orderings of certain tangents of input objects for this purpose. The minimum area problem uses a similar dynamic program, see the full version for its details.

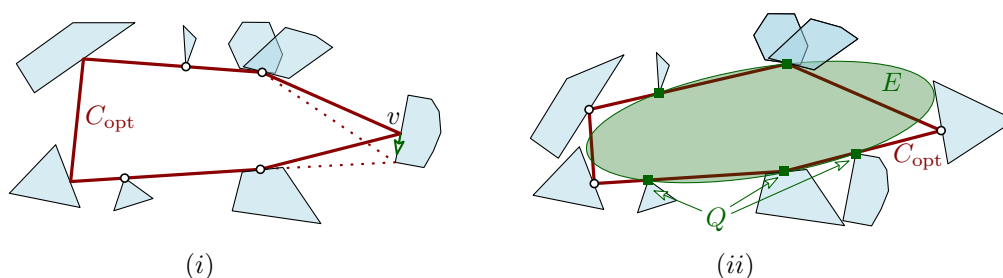
Finally, in order to present our exact algorithm in Section 4, we need to modify our dynamic program to deal with subproblems where the vertices of a convex chain do not come from a discretized set. In such cases, we have to find the order in which the objects of \mathcal{O} are visited by the chain. We are able to prove a specific ordering only in the case when the objects are line segments. The order then allows us to invoke the algorithm of Dror et al. [2] in a black-box manner.

2 Locating an optimal solution

The algorithms to be presented in subsequent sections need to approximately know the size and location of a smallest intersecting polygon. We use an algorithm from [4] to locate the minimum-perimeter intersecting polygon. With respect to the minimum-area intersecting polygon we prove that either there is a solution with a constant number of vertices (that can be computed with a different algorithm), or it is sufficient to consider polygons whose vertices are from a grid which comes from a polynomial collection of different grids.

Locating the minimum-perimeter optimum. For the minimum-perimeter intersecting polygon of a set \mathcal{O} of convex objects, Dumitrescu and Jiang [4] present an algorithm $A1$ that, for a given $\varepsilon_1 > 0$, outputs a rectangle R intersecting all input objects \mathcal{O} and with perimeter at most $\frac{4}{\pi}(1 + \varepsilon_1)\text{OPT}$. At a high level, $A1$ guesses an orientation of the rectangle among $\lceil \frac{\pi}{4\varepsilon_1} \rceil$ many discrete orientations and then uses a linear program to identify the smallest perimeter rectangle of that orientation that intersects \mathcal{O} . In [4] it is described how Algorithm $A1$ is used to locate an optimal solution if the input objects are convex polygons. In particular, for any $\varepsilon > 0$ running $A1$ with $\varepsilon_1 = \frac{\varepsilon}{2 + \varepsilon}$ gives a rectangle R . Let σ be the square that is concentric and parallel to R and has a side length of $3 \cdot \text{per}(R)$. Then the following holds.

► **Lemma 4** (Lemma 3 in [4]). *Suppose that $\text{per}(R) \geq (1 + \varepsilon)\text{OPT}$. Then there is an optimum polygon C_{opt} that is covered by σ .*



■ **Figure 1** (i) If v and adjacent sides of C_{opt} are disjoint from X , then we can slide v without increasing the area of C_{opt} . (ii) The points of Q (green) and their circumscribed ellipse E .

Algorithm A1 needs to solve $O(1/\varepsilon)$ many linear programs with $O(n)$ variables and $O(n)$ constraints each. Thus R and σ can be found in $O(T_{\text{LP}}(n)/\varepsilon)$ time, where $O(T_{\text{LP}}(n))$ is the running time of an LP solver with $O(n)$ variables and $O(n)$ constraints. The state-of-the-art LP solver by Jiang et al. [8] achieves a running time better than $O(n^{2.373})$. Lemma 4 directly implies that if $\text{per}(R) \geq (1 + \varepsilon)\text{per}(C_{\text{opt}})$, then

$$\text{diam}(C_{\text{opt}}) \leq \text{diam}(\sigma) = 3\sqrt{2}\text{per}(R) \leq 3\sqrt{2}\frac{4}{\pi}(1 + \varepsilon)\text{per}(C_{\text{opt}}) = O(\text{diam}(C_{\text{opt}})). \quad (1)$$

The shape and location of the minimum-area optimum. For the rest of this section, let X denote the set of vertices in the planar arrangement given by \mathcal{O} .

► **Lemma 5.** *Let C_{opt} be a minimum-area intersecting polygon for the input \mathcal{O} that has the minimum number of vertices, and among such polygons has the maximum number of points from X on its boundary. Then for any vertex v of C_{opt} that is not in X , the relative interior of at least one side of C_{opt} adjacent to v contains a point of X .*

Proof. Suppose for a contradiction that $v \notin X$ and that the relative interior of the sides in C_{opt} adjacent to v are disjoint from X . Observe that v must be on the boundary of an input object, so it is in the relative interior of an edge e of an input object, see Fig. 1(i). Then there exists a vector parallel to e along which we can move v while fixing its neighboring vertices in C_{opt} , without increasing the area of C_{opt} . This movement can be continued until we hit a point in X , or the angle of the polygon becomes π at v_1 or v_2 . As a result, we end up with a feasible polygon S whose area is no greater than that of C_{opt} , and it has one less vertex or at least one more point of X on its boundary. This contradicts the properties of C_{opt} . ◀

The following lemma is more involved. Its full proof can be found in the full version.

► **Lemma 6.** *For any given set of input polygons \mathcal{O} and $0 < \varepsilon < 1$ there is an intersecting polygon C of area $(1 + \varepsilon)\text{OPT}$ which either has at most 8 vertices, or its vertices are in a rectangular grid G of size $O(1/\varepsilon^3) \times O(1/\varepsilon^3)$ where G belongs to a collection \mathcal{G} of grids that can be generated in polynomial time.*

Proof sketch. Let $Q = X \cap \partial C_{\text{opt}}$. By Lemma 5 we can show that $|Q| \geq \lceil |V(C_{\text{opt}})|/2 \rceil$, so if $|V| \geq 9$, then $|Q| \geq 4$. We guess the circumscribed ellipse of Q , and use an affinity on the entire instance and C_{opt} which transforms the ellipse into a circle E . Wlog. we assume that E is the unit circle centered at the origin. It is known that the disk with boundary $\frac{1}{2}E$ is covered by $\text{conv}(Q)$ and thus by C_{opt} .

Consider the division of ∂C_{opt} defined by the points of Q . A *section* of ∂C_{opt} between two consecutive points of Q is a *spike* if it has a vertex $v \in V(C_{\text{opt}})$ that is at distance $\Omega(1/\varepsilon)$ from the origin. One can show that if such a spike exists, then $\text{area}(C_{\text{opt}})$ is $\Omega(1/\varepsilon)$.

On the other hand, we can show that C_{opt} cannot have more than 2 spikes. We then claim that if C_{opt} has a spike, then there is a polygon on at most 8 vertices that has area at most $(1 + \varepsilon)\text{area}(C_{\text{opt}})$. This is because C_{opt} can be covered by the intersection of the “cones” defined by the spike(s), which defines a polygon C . We can show that the extra area in the intersection of the (at most two) spike cones is $O(1)$, thus $\text{area}(C) \leq (1 + \varepsilon)\text{area}(C_{\text{opt}})$, and C has at most 8 vertices, as desired.

If C_{opt} has no spikes, then all of its vertices are within distance $O(1/\varepsilon)$ from the origin. We define a fine grid G (that depends on the guessed ellipse E) in the radius $O(1/\varepsilon)$ square around the origin, and use standard arguments to show that the minimum area intersecting polygon whose vertices are from G has area at most $(1 + \varepsilon)\text{OPT}$. We can therefore return all the grids G for each guess of the circumscribed ellipse of Q . ◀

3 An FPTAS for the minimum-perimeter problem of convex objects in the plane

Let \mathcal{O} be a set of n convex objects in the plane for which we want to compute a minimum-perimeter convex intersecting polygon. We assume that \mathcal{O} cannot be stabbed by a single point – this is easy to test without increasing the total running time. Since a minimum-perimeter intersecting polygon is necessarily convex, we will from now on drop the adjective “convex” from our terminology. We do this even when referring to convex intersecting polygons that are not necessarily of minimum perimeter.

In the previous section we have seen that for any $\varepsilon > 0$, we can find a feasible rectangle R and a square σ with the following property: Either $\text{per}(R) \leq (1 + \varepsilon)\text{OPT}$, or $C_{\text{opt}} \subseteq \sigma$ with $\text{diam}(C_{\text{opt}}) = \Omega(1)\text{diam}(\sigma)$. Next we describe an algorithm that, given a parameter $\varepsilon > 0$ and a corresponding square σ , outputs an intersecting polygon $C^* \subseteq \sigma$ for \mathcal{O} such that if $\text{per}(R) \geq (1 + \varepsilon)\text{per}(C_{\text{opt}})$ then $\text{per}(C^*) \leq (1 + \varepsilon)\text{OPT}$, where $\text{OPT} = \text{per}(C_{\text{opt}})$ (cf. Lemma 4 and Equation 1). Finally, we output either R or C^* , whichever has smaller perimeter.

Our algorithm starts by partitioning σ into a regular grid $G(\sigma)$ of $O(1/\varepsilon^2)$ cells of edge length at most $(\varepsilon/8) \cdot \text{OPT}$. We say that a convex polygon is a *grid polygon* if its vertices are grid points from $G(\sigma)$. The following observation is standard, but a full proof can be found in the full version.

► **Observation 7.** *Suppose σ contains an optimal solution C_{opt} . Let $C(\sigma)$ be a minimum-perimeter grid polygon that is an intersecting polygon for \mathcal{O} . Then $\text{per}(C(\sigma)) \leq (1 + \varepsilon) \cdot \text{OPT}$.*

Next we describe an algorithm to compute a minimum-perimeter grid polygon $C(\sigma)$ that is an intersecting polygon for \mathcal{O} .

First, we “guess” the lexicographically smallest vertex v_{bot} of $C(\sigma)$, see Figure 2(i). We can guess v_{bot} in $O(1/\varepsilon^2)$ different ways. For each possible guess we will find the best solution (if it exists), and then we report the best solution found over all guesses.

Now consider a fixed guess for the lexicographically smallest vertex v_{bot} of $C(\sigma)$. With a slight abuse of notation we will use $C(\sigma)$ to denote a minimum-perimeter grid polygon that is an intersecting polygon of \mathcal{O} and that has v_{bot} as lexicographically smallest vertex. (If the polygon $C(\sigma)$ does not exist, the algorithm described below will detect this.) We will compute $C(\sigma)$ by dynamic programming.

The vertices of $C(\sigma)$ are grid points in the region $h^+ \setminus \rho_0$, where h^+ is the closed half-plane above the horizontal line through v_{bot} and ρ_0 is the horizontal ray emanating from v_{bot} and pointing to the left. Let V be the set of such grid points, excluding v_{bot} . We first order the points from V in angular order around v_{bot} . More precisely, for a point $v \in V$, let $\phi(v)$ denote

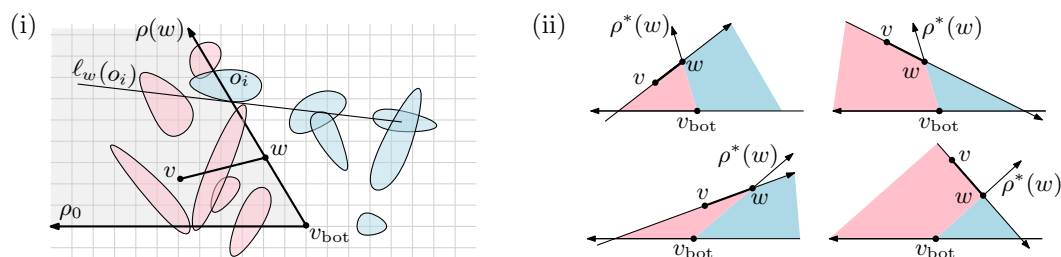


Figure 2 (i) The wedge defined by ρ_0 and $\rho(w)$ is shown in light grey. Objects in $\mathcal{O}(v, w)$ are red, objects in $\mathcal{O} \setminus \mathcal{O}(v, w)$ are blue. (ii) The partial solution $C(\Gamma)$ must be contained in the red region, while $C \setminus C(\Gamma)$ must lie in the blue region. There are four situations, depending on whether the angle between ρ_0 and $\rho(w)$ is acute or not, and whether the line containing vw intersects ρ_0 or not.

the angle over which we have to rotate ρ_0 in clockwise direction until we hit v . For two points $v, w \in V$, we write $v \prec w$ if $\phi(v) < \phi(w)$. Let $V^+ := V \cup \{v_{\text{bot}}, \bar{v}_{\text{bot}}\}$, where \bar{v}_{bot} is a copy of v_{bot} , and define $v_{\text{bot}} \prec v$ and $v \prec \bar{v}_{\text{bot}}$ for all $v \in V$. The copy \bar{v}_{bot} serves to distinguish the start and the end vertex of the clockwise circular sequence of vertices of $C(\sigma)$. Note that if $v_{\text{bot}}, v_1, \dots, v_k, \bar{v}_{\text{bot}}$ denotes this circular sequence then $v_{\text{bot}} \prec v_1 \prec \dots \prec v_k \prec \bar{v}_{\text{bot}}$ (since $C(\sigma)$ will never have two vertices that make the same angle with v_{bot}).

We now describe our dynamic-programming algorithm. Consider a polyline from v_{bot} to some point $v \in V$. We say that this polyline is a *convex chain* if, together with the line segment vv_{bot} , it forms a convex polygon. We denote the convex polygon induced by such a chain Γ by $C(\Gamma)$. The problem we now wish to solve is as follows:

Compute a minimum-length convex chain Γ^* from v_{bot} to \bar{v}_{bot} such that $C(\Gamma^*)$ is an intersecting polygon for \mathcal{O} .

Our dynamic-programming algorithm uses the partial order \prec defined above. We now want to define a subproblem for each point $v \in V^+$, which is to find the “best” chain Γ_v ending at v . For this to work, we need to know which objects from \mathcal{O} should be covered by the partial solution $C(\Gamma_v)$. This is difficult, however, because objects that intersect the ray from v_{bot} and going through v could either be intersected by $C(\Gamma_v)$ or by the part of the solution that comes after v . To overcome this problem we let the subproblems be defined by the last edge on the chain, instead of by the last vertex. This way we can decide which objects should be covered by a partial solution, as explained next.

Consider a convex chain from v_{bot} to a point $w \in V^+$ whose last edge is vw . Let $\rho(w)$ be the ray emanating from v_{bot} in the direction of w , and let $\rho^*(w)$ be the part of the ray starting at w . For $w = \bar{v}_{\text{bot}}$ we define $\rho(w)$ to be the horizontal ray emanating from v_{bot} and going to the right, and we define $\rho^*(w) = \rho(w)$. For an object $o_i \in \mathcal{O}$ that intersects $\rho^*(w)$, let $\ell_w(o_i)$ be a line that is tangent to o_i at the first intersection point of $\rho^*(w)$ with o_i . We now define the set $\mathcal{O}(v, w)$ to be the subset of objects $o_i \in \mathcal{O}$ such that one of the following conditions is satisfied; see also Figure 2(i).

- (i) o_i intersects the wedge defined by ρ_0 and $\rho(w)$, but not $\rho(w)$ itself; or
- (ii) o_i intersects $v_{\text{bot}}w$; or
- (iii) o_i intersects $\rho^*(w)$ but not $v_{\text{bot}}w$, and the tangent line $\ell_w(o_i)$ intersects the half-line containing vw and ending at w .

The next lemma shows that we can use the sets $\mathcal{O}(v, w)$ to define our subproblems.

► **Lemma 8.** *Let C be any convex polygon that is an intersecting polygon for \mathcal{O} and that has v_{bot} as lexicographically smallest vertex and vw as one of its edges. Let Γ_w be the part of ∂C from v_{bot} to w in clockwise direction. Then all objects in $\mathcal{O}(v, w)$ intersect $C(\Gamma_w)$ and all objects in $\mathcal{O} \setminus \mathcal{O}(v, w)$ intersect $C \setminus C(\Gamma_w)$.*

9:8 Computing Smallest Convex Intersecting Polygons

We can now state our dynamic program. To this end we define, for two points $v, w \in V^+$ with $v \prec w$, a table entry $A[v, w]$ as follows.

$A[v, w] :=$ the minimum length of a convex chain Γ from v_{bot} to w whose last edge is vw and such that all objects in $\mathcal{O}(v, w)$ intersect $C(\Gamma)$,

where the minimum is ∞ if no such chain exists. Lemma 8 implies the following.

► **Observation 9.** *Let Γ^* be a shortest convex chain from v_{bot} to \bar{v}_{bot} such that $C(\Gamma^*)$ is an intersecting polygon for \mathcal{O} . Then $\text{length}(\Gamma^*) = \min\{A[v, \bar{v}_{\text{bot}}] : v \in V \text{ and } \mathcal{O}(v, \bar{v}_{\text{bot}}) = \mathcal{O}\}$.*

Hence, if we can compute all table entries $A[v, w]$ then we have indeed solved our problem. (The lemma only tells us something about the value of an optimal solution, but given the table entries $A[v, w]$ we can compute the solution itself in a standard way.)

The entries $A[v, w]$ can be computed using the following lemma. Define $\Delta(v_{\text{bot}}, v, w)$ to be the triangle with vertices v_{bot}, v, w .

► **Lemma 10.** *Let $v, w \in V^+$ with $v \prec w$. Let $V(v, w)$ be the set of all points $u \in V \cup \{v_{\text{bot}}\}$ with $u \prec v$ such that u lies below the line $\ell(v, w)$ through v and w and such that all objects in $\mathcal{O}(v, w) \setminus \mathcal{O}(u, v)$ intersect $\Delta(v_{\text{bot}}, v, w)$. Then*

$$A[v, w] = \begin{cases} |v_{\text{bot}}w| & \text{if } v = v_{\text{bot}} \text{ and all objects in } \mathcal{O}(v, w) \text{ intersect } v_{\text{bot}}w \\ \infty & \text{if } v = v_{\text{bot}} \text{ and not all objects in } \mathcal{O}(v, w) \text{ intersect } v_{\text{bot}}w \\ |vw| + \min_{u \in V(v, w)} A[u, v] & \text{otherwise} \end{cases}$$

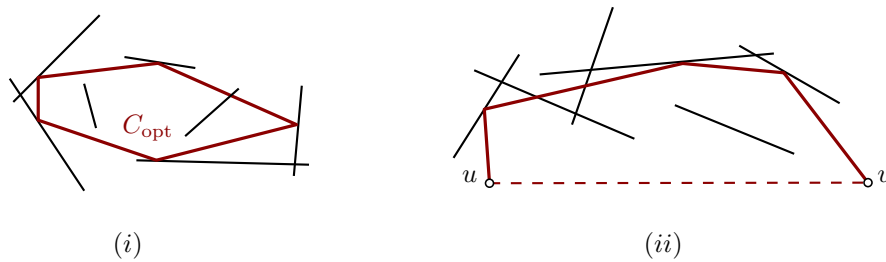
Putting everything together, we can finish the proof of Theorem 1.

Proof of Theorem 1. We first use algorithm A1 from [4] to compute the rectangle R and the square σ , which as discussed can be done in $O(n^{2.373}/\varepsilon)$ time. For a square σ we guess the vertex v_{bot} in $O(1/\varepsilon^2)$ different ways.

For each guess we run the dynamic-programming algorithm described above. There are $O(1/\varepsilon^4)$ entries $A[u, v]$ in the dynamic-programming table. The most time-consuming computation of a table entry is in the third case of Lemma 10. Here we need to compute the set $\mathcal{O}(v, w)$, which can be done in $O(n)$ time by checking every $o_i \in \mathcal{O}$. For each of the $O(1/\varepsilon^2)$ points with $u \prec v$ such that u lies below the line $\ell(v, w)$ we then check in $O(n)$ time if all objects in $\mathcal{O}(v, w) \setminus \mathcal{O}(u, v)$ intersect $\Delta(v_{\text{bot}}, v, w)$, so that we can compute $A[v, w]$. Hence, computing $A[v, w]$ takes $O(n/\varepsilon^2)$ time, which implies that the whole dynamic program needs $O(n/\varepsilon^6)$ time.

Thus the algorithm takes $O(n^{2.373}/\varepsilon) + O(1/\varepsilon^2) \cdot O(n/\varepsilon^6) = O(n^{2.373}/\varepsilon + n/\varepsilon^8)$ time. ◀

► **Remark 11.** Although Theorem 1 is stated only for the case where \mathcal{O} is a set of convex polygons, it is not too hard to extend it to other convex objects, for example disks: one just needs to replace the approximate rectangle-finding linear program of Dumitrescu and Jiang [4] with some other polynomial-time algorithm to find an (approximate) minimum perimeter intersecting rectangle in each of the $O(1/\varepsilon)$ orientations.



■ **Figure 3** (i) Subroutine I computes a minimum perimeter intersecting polygon whose vertices are not segment endpoints. (ii) Subroutine II computes a polygon with fixed edge uv for some (sub)set of segments. With the exception of u and v , the polygon's vertices are not allowed to be segment endpoints.

4 An exact algorithm for the minimum-perimeter intersecting polygon of segments

We describe an exact algorithm to compute a minimum-perimeter intersecting object for a set \mathcal{O} of line segments¹ in the plane. Consider an optimal solution C_{opt} , and let Y be the set of all endpoints of the segments in \mathcal{O} . The exact algorithm is based on two subroutines for the following two problems. As before, whenever we talk about intersecting polygons, we implicitly require them to be convex. Figure 3 illustrates the polygons computed by these subroutines.

- *Subroutine I*: If \mathcal{O} admits a minimum-perimeter intersecting polygon with none of its vertices being in Y , then compute such a polygon. Otherwise compute a feasible intersecting polygon, or report $+\infty$.
- *Subroutine II*: Given two points $u, v \in \mathbb{R}^2$ and a subset $\mathcal{O}' \subseteq \mathcal{O}$, decide if \mathcal{O}' admits a minimum-perimeter intersecting polygon that has uv as one of its edges and none of whose other vertices belongs to Y and, if so, compute a minimum-perimeter such intersecting polygon. If no such minimum-perimeter intersecting polygon exists, report $+\infty$. Note that we allow $u = v$, in which case the edge uv degenerates to a point.

In the full version we show:

► **Theorem 12.** *There exist exact algorithms for Subroutine I and Subroutine II that run in time $O(n^6 \log n)$ and $O(n^3 \log n)$, respectively.*

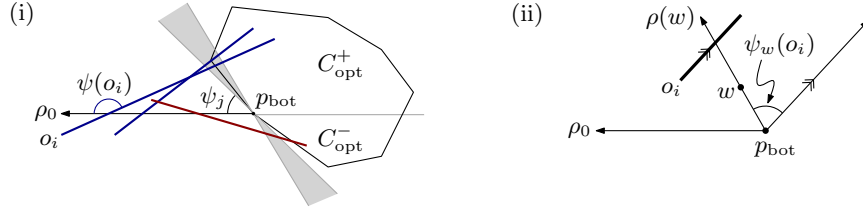
Setting the stage for the dynamic program

With Subroutine I available, it remains to find the minimum-perimeter intersecting polygon that has at least one vertex from Y . To this end we will develop an algorithm that, for a given point $p_{\text{bot}} \in Y$, finds a minimum-perimeter convex intersecting polygon that has p_{bot} as a vertex (if it exists). We will run this algorithm for each choice of $p_{\text{bot}} \in Y$.

Let $p_{\text{bot}} \in Y$ be given, and assume that \mathcal{O} admits an intersecting polygon that has p_{bot} as a vertex (this can be tested in $O(n \log n)$ time). In contrast to Section 3, p_{bot} need not be the lexicographically smallest point and our algorithm therefore relies on guessing the

¹ Although we describe our algorithm for non-degenerate segments, all our arguments also work if some (or all) of the segments are in fact lines, rays, or even points.

9:10 Computing Smallest Convex Intersecting Polygons



■ **Figure 4** (i) The grey double wedge indicates the region containing the tangent at p_{bot} . (Note that then there must be more segments than the blue and red segments that are shown. In particular, there must be segments parallel to the lines delimiting the double wedge.) The blue segments are in $\mathcal{O}(\rho_0)^+$ so they must be intersected by C_{opt}^+ , while the red segment is in $\mathcal{O}(\rho_0)^-$ and must be intersected by C_{opt}^- . (ii) The definition of $\psi_w(o_i)$.

orientation of a line tangent to C_{opt} at p_{bot} . More specifically, we are able to find a set $\Psi = \{\psi_1, \psi_2, \dots\}$, of $O(n)$ possible angles with $\psi_j < \psi_{j+1}$ for all $0 \leq j \leq |\Psi|$ (where we set $\psi_0 := 0$ and $\psi_{|\Psi|+1} = \pi$) and reduce the problem to:

Given a point p_{bot} and value j with $0 \leq j \leq |\Psi|$, find a minimum-perimeter intersecting polygon C_{opt} for \mathcal{O} such that

- p_{bot} is a vertex of C_{opt} ,
- the horizontal ray ρ_0 going from p_{bot} to the left does not intersect C_{opt} ,
- C_{opt} has a tangent line ℓ at p_{bot} such that the clockwise angle from ρ_0 to ℓ lies in the range $[\psi_j, \psi_{j+1}]$.

In the above, each angle ψ_j corresponds to the angle over which we have to rotate ρ_0 clockwise so that it becomes parallel with some segment $o_j \in \mathcal{O}$. An extensive description of this reduction can be found in the full version.

The dynamic program

We will now develop our dynamic program for the problem that we just stated, for a given point $p_{\text{bot}} \in Y$, a ray ρ_0 , and range $[\psi_j, \psi_{j+1}]$; see Figure 4(i).

Let $\mathcal{O}(\rho_0) \subseteq \mathcal{O}$ denote the set of segments that intersect the ray ρ_0 , and let ℓ_0 be the line containing ρ_0 . The line ℓ_0 may split the optimal solution C_{opt} into two parts: a part C_{opt}^+ above ℓ_0 and a part C_{opt}^- below ℓ_0 . Let $\psi(o_i)$ denote the angle over which we have to rotate ρ_0 in clockwise direction until it becomes parallel to o_i . Since we have fixed the range of the tangent at p_{bot} to lie in the range $[\psi_j, \psi_{j+1}]$, we can split $\mathcal{O}(\rho_0)$ into two subsets, $\mathcal{O}(\rho_0)^+ := \{o_i \in \mathcal{O}(\rho_0) : \psi(o_i) \geq \psi_{j+1}\}$ and $\mathcal{O}(\rho_0)^- := \{o_i \in \mathcal{O}(\rho_0) : \psi(o_i) \leq \psi_j\}$.

Note that $\mathcal{O}(\rho_0) = \mathcal{O}(\rho_0)^+ \cup \mathcal{O}(\rho_0)^-$, because ψ_j and ψ_{j+1} are consecutive angles in Ψ . Because the orientation of the tangent at p_{bot} lies in the range $[\psi_j, \psi_{j+1}]$, we know that the segments in $\mathcal{O}(\rho_0)^+$ must be intersected by C_{opt}^+ , while the segments in $\mathcal{O}(\rho_0)^-$ must be intersected by C_{opt}^- ; see Figure 4(i). Intuitively, the segments in $\mathcal{O}(\rho_0)^+$ must be intersected by “the initial part” of C_{opt} , while the segments in $\mathcal{O}(\rho_0)^-$ are intersected by “the later part”. We will use this when we define the subproblems in our dynamic program.

In Section 3 we defined subproblems for pairs of grid points v, w . The goal of such a subproblem was to find the minimum-length convex chain Γ such that $C(\Gamma)$ intersects a certain subset $\mathcal{O}(v, w)$ and whose last edge is vw . The fact that we knew the last edge vw was crucial to define the set $\mathcal{O}(v, w)$, since the slope of vw determined which objects should be intersected by $C(\Gamma)$. In the current setting this does not work: we could define a subproblem for pairs $v, w \in Y$, but “consecutive” vertices v, w from Y along Γ are now connected by a

polyline Z_{vw} whose inner vertices are disjoint from Y . The difficulty is that the polyline Z_{vw} depends on the segments that need to be intersected by $C(\Gamma)$. Hence, there is a cyclic dependency between the set of segments to be intersected by $C(\Gamma)$ (which depends on the slope of zw , where z is the vertex of Z_{vw} preceding w) and the vertex z (which depends on the segments that need to be intersected by $C(\Gamma)$). We overcome this problem as follows.

Similarly to the previous section, we call a polyline Γ from p_{bot} to some point $v \in Y$ a *convex chain* if, together with the line segment vp_{bot} , it forms a convex polygon. We denote this polygon by $C(\Gamma)$. Consider a convex chain Γ_w ending in a point $w \in Y$. Let $\rho(w)$ denote the ray from p_{bot} through w and let $\rho^*(w)$ be the part of this ray starting at w . Let $\mathcal{O}(w)$ be the set of input segments that intersect $\rho^*(w)$. Of those segments, $C(\Gamma_w)$ must intersect the ones such that the line $\ell(o_i)$ containing² the segment o_i intersects the half-line containing zw and ending at w , where z is the (unknown) vertex preceding w . For a segment $o_i \in \mathcal{O}(w)$, let $\psi_w(o_i)$ be the angle over which we have to rotate $\rho(w)$ in clockwise direction to make it parallel to o_i ; see Figure 4(ii). Let $\Psi(w) = \{\psi_1, \psi_2, \dots\}$, for all $1 \leq j < |\Psi(w)|$, be the sorted set of (distinct) angles $\psi_w(o_i)$ defined by the segments in $\mathcal{O}(w)$. For an index j with $1 \leq j \leq |\Psi(w)|$, define $\mathcal{O}(w, j) := \{o_i \in \mathcal{O}(w) : \psi_w(o_i) \leq \psi_j\}$, and define $\mathcal{O}(w, 0) = \emptyset$. We call $\mathcal{O}(w, j)$ a *prefix set*. The key observation is that $C(\Gamma_w)$ must intersect the segments from some prefix set $\mathcal{O}(w, j)$, where j depends on the unknown vertex z preceding w . So our dynamic program will try all possible prefix sets $\mathcal{O}(w, j)$, and make sure that subproblems are combined in a consistent manner.

We now have everything in place to describe our dynamic-programming table. It consists of entries $A[w, j]$, where w ranges over all points in Y , and j ranges over all values for which $\mathcal{O}(w, j)$ is defined. For convenience add two special entries, $A[p_{\text{bot}}, 0]$ and $A[\bar{p}_{\text{bot}}, 0]$; the former will serve as the base case, and the latter will contain (the value of) the final solution. Note that these are the only ones for p_{bot} and \bar{p}_{bot} , and that we have at most $|Y| \cdot n = O(n^2)$ entries. We define the set $\mathcal{O}^*(w, j)$ of segments to be covered in a subproblem.

For a point $w \in Y$, the set $\mathcal{O}^*(w, j)$ consists of the segments $o_i \in \mathcal{O}$ that satisfy one of the following conditions:

- (i) o_i intersects the clockwise wedge from ρ_0 to $\rho(w)$ – note that this wedge need not be convex – but not $\rho(w)$ itself, and $o_i \notin \mathcal{O}^-(\rho_0)$; or
- (ii) o_i intersects $p_{\text{bot}}w$; or
- (iii) $o_i \in \mathcal{O}(w, j)$.

Furthermore, $\mathcal{O}^*(p_{\text{bot}}, 0) := \emptyset$ and $\mathcal{O}^*(\bar{p}_{\text{bot}}, 0) := \mathcal{O}$.

We would like now to define $A[w, j]$ to be the minimum length of a convex chain Γ from p_{bot} to w such that all objects in $\mathcal{O}^*(w, j)$ intersect $C(\Gamma)$. There is, however, a technicality to address: the minimum-perimeter polygon that intersects all segments from \mathcal{O} need not be convex when we require it to have p_{bot} as a vertex. Such a non-convex polygon cannot be the final solution – if the optimum for a given choice of p_{bot} is non-convex, then p_{bot} was not the correct choice – but it makes a clean definition of our subproblems awkward. Therefore, instead of first defining the subproblems and then giving the recursive formula, we will immediately give the recursive formula and then prove that it computes what we want.

For two points $v, w \in Y^+$ (where $Y^+ = Y \cup \{\bar{p}_{\text{bot}}\}$) with $v \prec w$ and a set $\mathcal{O}' \subseteq \mathcal{O}$, let $L(v, w, \mathcal{O}')$ be the minimum length of a convex chain Γ from v to w such that the convex polygon defined by Γ and vw is an intersecting set for \mathcal{O}' and all inner vertices of Γ are disjoint from Y . Recall that we can compute $L(v, w, \mathcal{O}')$ using subroutine II. As before, let $\Delta(p_{\text{bot}}, v, w)$ denote the triangle with vertices p_{bot}, v, w .

² Since the input objects are now segments, the tangent line $\ell_w(o_i)$ is just the line containing o_i .

9:12 Computing Smallest Convex Intersecting Polygons

► **Definition 13.** Let $w \in Y^+$ and j be a value for which $\mathcal{O}[w, j]$ is defined. Thus $0 \leq j \leq |\Psi(w)|$, where we set $|\Psi(w)| := 0$ for $w \in \{p_{\text{bot}}, \bar{p}_{\text{bot}}\}$. For $v \prec w$ and $0 \leq j' \leq |\Psi(v)|$, let

$$\mathcal{O}^*(w, j, v, j') := \mathcal{O}^*(w, j) \setminus \left(\mathcal{O}^*(v, j') \cup \{o_i \in \mathcal{O} : o_i \text{ intersects } \Delta(p_{\text{bot}}, v, w)\} \right)$$

and define

$$A[w, j] := \begin{cases} 0 & \text{if } w = p_{\text{bot}} \\ \min_{\substack{v \prec w \\ 0 \leq j' \leq |\Psi(v)|}} L(v, w, \mathcal{O}^*(w, j, v, j')) + A[v, j'] & \text{otherwise.} \end{cases}$$

The next lemma implies that the table entry $A[\bar{p}_{\text{bot}}, 0]$ defined by this recursive formula gives us what we want. Part (a) implies that $A[\bar{p}_{\text{bot}}, 0]$ will never return a value that is too small, while part (b) implies that for the correct choice of p_{bot} and range of orientations for the tangent to C_{opt} at p_{bot} , the entry $A[\bar{p}_{\text{bot}}, 0]$ gives us (the value of) the optimal solution.

► **Lemma 14.** Consider the table entry $A[\bar{p}_{\text{bot}}, 0]$ defined by Definition 13 for a given point p_{bot} and range $[\psi_i, \psi_{i+1}]$.

- (a) There exists a convex intersecting polygon for \mathcal{O} of perimeter at most $A[\bar{p}_{\text{bot}}, 0]$.
- (b) If p_{bot} is a vertex of the minimum-perimeter convex intersecting polygon C_{opt} for \mathcal{O} , and ρ_0 does not intersect C_{opt} , and there is a tangent line ℓ at p_{bot} whose orientation is in the range $[\psi_i, \psi_{i+1}]$, then $\text{per}(C_{\text{opt}}) = A[\bar{p}_{\text{bot}}, 0]$.

Putting everything together

Lemma 14 implies that after solving the dynamic programs for all choices of p_{bot} and the range $[\psi_i, \psi_{i+1}]$, we have found the minimum perimeter intersecting set for \mathcal{O} . (Computing the intersecting set itself, using the relevant dynamic-program table, is then routine.) This leads to the proof of Theorem 3.

Proof of Theorem 3. The number of dynamic programs solved is $O(|Y| \cdot n) = O(n^2)$. The dynamic-programming tables have $O(n^2)$ entries. Computing an entry takes $O(|Y| \cdot n) = O(n^2)$ calls to Subroutine II, at $O(n^3 \log n)$ time each. The dynamic programs thus take $O(n^2) \cdot O(n^2) \cdot O(n^2) \cdot O(n^3 \log n) = O(n^9 \log n)$ time. If the optimal solution does not go through any point of Y , then by Theorem 12 it will be found in $O(n^6 \log n)$ time. The optimum of these two algorithms is the global optimum. ◀

5 Conclusion

We gave fully polynomial time approximation schemes for the minimum perimeter and minimum area convex intersecting polygon problems for convex polygons. Additionally, we developed a polynomial-time algorithm for the minimum perimeter problem of segments.

It is likely that the running times of our algorithms can be improved further. One could also try to generalize the set of objects, for example, adapting the minimum area algorithm to arbitrary convex objects. We propose the following open questions for further study.

- Is there a polynomial-time exact algorithm for the minimum area convex intersecting polygon of segments?
- Is there a polynomial-time exact algorithm for minimum perimeter or minimum area convex intersecting polygon of convex polygons, or are these problems NP-hard?

- Is there a polynomial-time approximation scheme for the minimum volume or minimum surface area convex intersecting polytope of convex polytopes in \mathbb{R}^3 ? Can we at least approximate the diameter of the optimum solution to these problems?

It would be especially interesting to see an NP-hardness proof for minimum volume or surface area convex intersecting set of convex objects in higher dimensions.

References

- 1 Antonios Antoniadis, Krzysztof Fleszar, Ruben Hoeksma, and Kevin Schewior. A PTAS for Euclidean TSP with hyperplane neighborhoods. *ACM Trans. Algorithms*, 16(3):38:1–38:16, 2020.
- 2 Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph SB Mitchell. Touring a sequence of polygons. In *STOC 2003: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 473–482, 2003.
- 3 Adrian Dumitrescu. The traveling salesman problem for lines and rays in the plane. *Discrete Mathematics, Algorithms and Applications*, 4(04):1250044, 2012.
- 4 Adrian Dumitrescu and Minghui Jiang. Minimum-perimeter intersecting polygons. *Algorithmica*, 63(3):602–615, 2012. doi:10.1007/s00453-011-9516-3.
- 5 Ray A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information processing letters*, 2(1):18–21, 1973.
- 6 Ahmad Javad, Ali Mohades, Mansoor Davoodi, and Farnaz Sheikhi. Convex hull of imprecise points modeled by segments in the plane, 2010.
- 7 Yiyang Jia and Bo Jiang. The minimum perimeter convex hull of a given set of disjoint segments. In *International Conference on Mechatronics and Intelligent Robotics*, pages 308–318. Springer, 2017.
- 8 Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. A faster algorithm for solving general lps. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing 2021*, pages 823–832. ACM, 2021. doi:10.1145/3406325.3451058.
- 9 Marc J. van Kreveld and Maarten Löffler. Approximating largest convex hulls for imprecise points. *J. Discrete Algorithms*, 6(4):583–594, 2008. doi:10.1016/j.jda.2008.04.002.
- 10 Maarten Löffler and Marc J. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010. doi:10.1007/s00453-008-9174-2.
- 11 Franco P. Preparata and Se June Hong. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20(2):87–93, 1977.
- 12 Xuehou Tan. The touring rays and related problems. *Theoretical Computer Science*, 2021.
- 13 Csaba D. Tóth, Joseph O'Rourke, and Jacob E Goodman. *Handbook of discrete and computational geometry*. CRC press, 2017.

The Price of Hierarchical Clustering

Anna Arutyunova ✉

Universität Bonn, Germany

Heiko Röglin ✉

Universität Bonn, Germany

Abstract

Hierarchical Clustering is a popular tool for understanding the hereditary properties of a data set. Such a clustering is actually a sequence of clusterings that starts with the trivial clustering in which every data point forms its own cluster and then successively merges two existing clusters until all points are in the same cluster. A hierarchical clustering achieves an approximation factor of α if the costs of each k -clustering in the hierarchy are at most α times the costs of an optimal k -clustering. We study as cost functions the maximum (discrete) radius of any cluster (k -center problem) and the maximum diameter of any cluster (k -diameter problem).

In general, the optimal clusterings do not form a hierarchy and hence an approximation factor of 1 cannot be achieved. We call the smallest approximation factor that can be achieved for any instance the *price of hierarchy*. For the k -diameter problem we improve the upper bound on the price of hierarchy to $3 + 2\sqrt{2} \approx 5.83$. Moreover we significantly improve the lower bounds for k -center and k -diameter, proving a price of hierarchy of exactly 4 and $3 + 2\sqrt{2}$, respectively.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Hierarchical Clustering, approximation Algorithms, k -center Problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.10

Related Version *Full Version*: <https://arxiv.org/abs/2205.01417> [4]

Funding This work has been supported by the German Research Association (DFG) grant RO 5439/1-1.

1 Introduction

Clustering is an ubiquitous task in data analysis and machine learning. In a typical clustering problem, the goal is to partition a set of objects into different clusters such that only similar objects belong to the same cluster. There are numerous ways how clustering can be modeled formally and many different models have been studied in the literature in the last decades. In many theoretical models, one assumes that the data comes from a metric space and that the desired number of clusters is given. Then the goal is to optimize some objective function like k -center, k -median, or k -means. In most cases the resulting optimization problems are NP-hard and hence approximation algorithms have been studied extensively.

One aspect of real-world clustering problems that is not captured by these models is that it is often already a non-trivial task to determine for a given data set the right or most reasonable number of clusters. One particularly appealing way to take this into account is hierarchical clustering. A hierarchical clustering of a data set is actually a sequence of clusterings, one for each possible number of clusters. It starts with the trivial clustering in which every data point forms its own cluster and then successively merges two existing clusters until all points are in the same cluster. This way for every possible number of clusters, a clustering is obtained. These clusterings help to understand the hereditary properties of the data and they provide information at different levels of granularity.



© Anna Arutyunova and Heiko Röglin;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 10; pp. 10:1–10:14



Leibniz International Proceedings in Informatics

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

While hierarchical clustering is successfully used in many applications, it is not as well understood from a theoretical point of view as the models in which the number of clusters is given as part of the input. One reason for this is that it is not obvious how the quality of a hierarchical clustering should be measured. A possibility that has been explored in the literature is to define the quality of a hierarchical clustering based on its worst level. To be precise, let (\mathcal{X}, d) be a metric space and $\mathcal{P} \subset \mathcal{X}$ a set of n points. Furthermore let $\mathcal{H} = (\mathcal{H}_n, \dots, \mathcal{H}_1)$ be a hierarchical clustering of \mathcal{P} , where \mathcal{H}_k denotes a k -clustering, i.e., a clustering with at most k non-empty clusters. Then \mathcal{H}_{k-1} arises from \mathcal{H}_k by merging some of the existing clusters. We assume that some objective function like k -center, k -median, or k -means is selected and denote by $\text{cost}(\mathcal{H}_k)$ the objective value of \mathcal{H}_k with respect to the selected objective function. Furthermore, let \mathcal{O}_k denote an optimal k -clustering and let $\text{cost}(\mathcal{O}_k)$ denote its objective value. Then we say that \mathcal{H} achieves an approximation factor of $\alpha \geq 1$ if $\text{cost}(\mathcal{H}_k) \leq \alpha \cdot \text{cost}(\mathcal{O}_k)$ for every k , assuming that cost is an objective that is to be minimized. In this work we consider the radius objective, which is well-known from the k -center problem. Here the cost is defined as the maximum radius of a cluster. Furthermore we consider the diameter objective, where the cost is defined as the maximum distance between any two points lying in the same cluster.

An α -approximation for small α yields a strong guarantee for the hierarchical clustering on every level. However, in general there do not exist optimal clusterings $\mathcal{O}_n, \dots, \mathcal{O}_1$ that form a hierarchy. So even with unlimited computational resources, a 1-approximation usually cannot be achieved. In the literature different algorithms for computing hierarchical clusterings with respect to different objective functions have been developed and analyzed. Dasgupta and Long [13] and Charikar et al. [8] initiated this line of research and presented both independently from each other an algorithm that computes efficiently an 8-approximate hierarchical clustering with respect to the radius and diameter objective. That is, for every level k , the maximal radius or diameter of any cluster in the k -clustering computed by their algorithms is at most 8 times the maximal radius or diameter in an optimal k -clustering. Inspired by [13], Plaxton [21] proposed a constant-factor approximation for the k -median and k -means objective. Later a general framework that also leads constant approximation guarantees for many objective functions including in particular k -median and k -means has been proposed by Lin et al. [19].

Despite these articles and other related work, which we discuss below in detail, many questions in the area of hierarchical clustering are not yet resolved. We find it particularly intriguing to find out which approximation factors can be achieved for different objectives. This question comes in two flavors depending on the computational resources available. Of course it is interesting to study which approximation factors can and cannot be achieved in polynomial time, assuming $P \neq NP$. Since in general there do not exist hierarchical clusterings that are optimal on each level, it is also interesting to study which approximation factors can and cannot be achieved in general without the restriction to polynomial-time algorithms.

For an objective function like radius or diameter we define its *price of hierarchy* as the smallest α such that for any instance there exists an α -approximate hierarchical clustering. Hence, the price of hierarchy is a measure for how much quality one has to sacrifice for the hierarchical structure of the clusterings.

Our main results are tight bounds for the price of hierarchy for the radius, discrete radius and diameter objective. Here the difference between radius and discrete radius lies in the choice of centers. For the radius objective we allow to choose the center of a cluster $C \subset \mathcal{P}$ from the whole metric space \mathcal{X} , while for the discrete radius objective the center must be contained in C itself. We will see that this has an impact on the price of hierarchy.

For all three objectives the algorithms in [13, 8] compute an 8-approximate hierarchical clustering in polynomial time. Until recently this was also the best known upper bound for the price of hierarchy in the literature for hierarchical radius and diameter. For discrete radius, Großwendt [17] shows an upper bound for the price of hierarchy of 4. The best known lower bounds are 2, proven by Das and Kenyon-Mathieu [11] for diameter and by Großwendt [17] for (discrete) radius. We improve the framework in [19] for radius and diameter and show an upper bound on the price of hierarchy of $3 + 2\sqrt{2} \approx 5.83$. The upper bound of $3 + 2\sqrt{2}$ for the radius was also recently proved by Bock [5] in independent work. However our main contribution lies in the design of clustering instances to prove a lower bound of 4 for discrete radius and $3 + 2\sqrt{2}$ for radius and diameter.

Related work. Gonzales [14] presents a simple and elegant incremental algorithm for k -center. The algorithm exhibits the following nice property: given a set \mathcal{P} which has to be clustered, it returns an ordering of the points, such that the first k points constitute the centers of the k -center solution, and this solution is a 2-approximation for every $1 \leq k \leq |\mathcal{P}|$. However the resulting clusterings are usually not hierarchically compatible. Dasgupta and Long [13] use the ordering computed by Gonzales' algorithm to compute a hierarchical clustering. The authors present an 8-approximation for the objective functions (discrete) radius and diameter. In an independent work Charikar et al. [8] also present an 8-approximation for the three objectives which outputs the same clustering as the algorithm in [13] under some reasonable conditions [11]. In a recent work, Mondal [20] gives a 6-approximation for hierarchical (discrete) radius. In the full version of this paper [4] we present an instance where this algorithm computes only a 7-approximation contradicting the claimed guarantee.

Plaxton [21] shows that a similar approach as in [13] yields a hierarchical clustering with constant approximation guarantee for the k -median and k -means objectives. Later a general framework for a variety of incremental and hierarchical problems was introduced by Lin et al. [19]. Their framework can be applied to compute hierarchical clusterings for any cost function which satisfies a certain nesting property, especially those of k -median and k -means. This yields a 20.71α -approximation for k -median and a 576β -approximation for k -means. Here $\alpha = 2.675$ and $\beta = 6.357$ are the currently best approximation guarantees for k -median [6] and k -means [2]. The algorithms presented in [8, 13, 19, 21] run in polynomial time. Unless $P=NP$ there is no polynomial-time α -approximation for $\alpha < 2$ for hierarchical (discrete) radius and diameter. For (discrete) radius this is an immediate consequence of the reduction from dominating set presented by [18]. A similar reduction from clique cover yields the statement for hierarchical diameter.

However even without time constraints it is not clear what approximation guarantee can be achieved for hierarchical clustering. It is easy to find examples, where the approximation guarantee of any hierarchical clustering for all three objectives is greater than one. Das and Kenyon-Mathieu [11] and Großwendt [17] present instances for diameter and (discrete) radius, where no hierarchical clustering has an approximation guarantee smaller than 2. On the other hand Großwendt [17] proves an upper bound of 4 on the approximation guarantee of hierarchical discrete radius by using the framework of Lin et al. [19]. In recent independent work Bock [5] improved the bound for hierarchical radius to $3 + 2\sqrt{2}$. While his approach is inspired by Dasgupta and Long [13], the resulting algorithm is similar to the algorithm we present in this paper as an improvement of [19].

Aside from the theoretical results, there also exist greedy heuristics, which are more commonly used in applications. One very simple bottom up, also called agglomerative, algorithm is the following: starting from the clustering where every point is separate, it

merges in every step the two clusters whose merge results in the smallest increase of the cost function. For (discrete) radius and diameter this algorithm is known as complete linkage and for the k -means cost this is Ward's method [23]. Ackermann et al. [1] analyze the approximation guarantee of complete linkage in the Euclidean space. They show an approximation guarantee of $O(\log(k))$ for all three objectives assuming the dimension of the Euclidean space to be constant. This was later improved by Großwendt and Röglin [15] to $O(1)$. In arbitrary metric spaces complete linkage does not perform well. There Arutyunova et al. [3] prove a lower bound of $\Omega(k)$ for all three objectives. For Ward's method Großwendt et al. [16] show an approximation guarantee of 2 under the strong assumption that the optimal clusters are well separated.

Recently other cost functions for hierarchical clustering were proposed, which do not compare to the optimal clustering on every level. Dasgupta [12] defines a new cost function for similarity measures and presents an $O(\alpha \log(n))$ -approximation for the respective problem. This was later improved to $O(\alpha)$ independently by Charikar and Chatziafratis [7] and Cohen-Addad et al. [9]. Here α is the approximation guarantee of sparsest cut. However Cohen-Addad et al. [9] prove that every hierarchical clustering is an $O(1)$ -approximation to the corresponding cost function for dissimilarity measures when the dissimilarity measure is a metric. A cost function more suitable for Euclidean spaces was developed by Wang and Moseley [22]. They prove that a randomly generated hierarchical clustering performs poorly for this cost function and show that bisecting k -means computes an $O(1)$ -approximation.

Our results. We define the *price of hierarchy* ρ_{cost} with respect to an objective function cost as the smallest number such that for every clustering instance there exists a hierarchical clustering which is a ρ_{cost} -approximation with respect to cost . Observe that the results [8, 11, 13, 17] imply that the price of hierarchy for radius and diameter is between 2 and 8 and for discrete radius between 2 and 4. We close these gaps and prove that the price of hierarchy for radius and diameter is exactly $3 + 2\sqrt{2}$ and for discrete radius exactly 4. Notice that this does not imply the existence of polynomial-time algorithms with approximation guarantee ρ_{cost} . Especially our algorithm which computes a $3 + 2\sqrt{2}$ -approximation for radius and diameter does not run in polynomial time. This is also the case for the $3 + 2\sqrt{2}$ -approximation for radius presented by Bock [5] in independent work. Our upper bound of $3 + 2\sqrt{2}$ can be achieved by a small improvement in the framework of Lin et al. [19]. However our most technically demanding contribution is the design of a clustering instance for every $\epsilon > 0$ such that every hierarchical clustering has approximation guarantee at least $3 + 2\sqrt{2} - \epsilon$ for radius and diameter and $4 - \epsilon$ for discrete radius. It requires a careful analysis of all possible hierarchical clusterings, which is highly non-trivial for complex clustering instances.

2 Preliminaries

A clustering instance $(\mathcal{X}, \mathcal{P}, d)$ consists of a metric space (\mathcal{X}, d) and a finite subset $\mathcal{P} \subset \mathcal{X}$. For a set (or cluster) $C \subset \mathcal{P}$ we denote by

$$\text{diam}(C) = \max_{p, q \in C} d(p, q)$$

the *diameter* of C . By $\text{rad}(C, c) = \max_{p \in C} d(c, p)$ we denote the radius of C with respect to a center $c \in \mathcal{X}$. This is the largest distance between c and a point in C . The *radius* of C is defined as the smallest radius of C with respect to a center $c \in \mathcal{X}$, i.e.,

$$\text{rad}(C) = \min_{c \in \mathcal{X}} \text{rad}(C, c)$$

while the *discrete radius* of C is defined as the smallest radius of C with respect to a center $c \in C$, i.e.,

$$\text{drad}(C) = \min_{c \in C} \text{rad}(C, c).$$

A k -clustering of \mathcal{P} is a partition of \mathcal{P} into at most k non-empty subsets. We consider three closely related clustering problems.

The k -diameter problem asks to minimize $\text{diam}(\mathcal{C}_k) = \max_{C \in \mathcal{C}_k} \text{diam}(C)$, i.e., the maximum diameter of a k -clustering \mathcal{C}_k . In the k -center problem we want to minimize the maximum radius $\text{rad}(\mathcal{C}_k) = \max_{C \in \mathcal{C}_k} \text{rad}(C)$, and in the discrete k -center problem we want to minimize the maximum discrete radius $\text{drad}(\mathcal{C}_k) = \max_{C \in \mathcal{C}_k} \text{drad}(C)$.

► **Definition 1.** Given an instance $(\mathcal{X}, \mathcal{P}, d)$, let $n = |\mathcal{P}|$. We call two clusterings \mathcal{C} and \mathcal{C}' of \mathcal{P} with $|\mathcal{C}| \geq |\mathcal{C}'|$ hierarchically compatible if for all $C \in \mathcal{C}$ there exists $C' \in \mathcal{C}'$ with $C \subset C'$. A hierarchical clustering of \mathcal{P} is a sequence of clusterings $\mathcal{H} = (\mathcal{H}_n, \dots, \mathcal{H}_1)$, such that

1. \mathcal{H}_i is an i -clustering of \mathcal{P}
2. for $1 < i \leq n$ the two clusterings \mathcal{H}_{i-1} and \mathcal{H}_i are hierarchically compatible.

For $\text{cost} \in \{\text{diam}, \text{rad}, \text{drad}\}$ let \mathcal{O}_i denote the optimal i -clustering with respect to cost . We say that \mathcal{H} is an α -approximation with respect to cost if for all $i = 1, \dots, n$ we have

$$\text{cost}(\mathcal{H}_i) \leq \alpha \cdot \text{cost}(\mathcal{O}_i).$$

Since optimal clusterings are generally not hierarchically compatible, there is usually no hierarchical clustering with approximation guarantee $\alpha = 1$. We have to accept that the restriction on hierarchically compatible clusterings comes with an unavoidable increase in the cost compared to an optimal solution.

► **Definition 2.** For $\text{cost} \in \{\text{diam}, \text{rad}, \text{drad}\}$ the price of hierarchy $\rho_{\text{cost}} \geq 1$ is defined as follows.

1. For every instance $(\mathcal{X}, \mathcal{P}, d)$, there exists a hierarchical clustering \mathcal{H} of \mathcal{P} that is a ρ_{cost} -approximation with respect to cost .
2. For any $\alpha < \rho_{\text{cost}}$ there exists an instance $(\mathcal{X}, \mathcal{P}, d)$, such that there is no hierarchical clustering of \mathcal{P} that is an α -approximation with respect to cost .

Thus ρ_{cost} is the smallest possible number such that for every clustering instance there is a hierarchical clustering with approximation guarantee ρ_{cost} .

3 An Upper Bound on the Price of Hierarchy

It is already known that the framework of Lin et al. [19] yields an upper bound of 4 on the price of hierarchy for the discrete radius [17]. This framework also yields upper bounds for the price of hierarchy for radius and diameter, which are not tight, however. We present an improved version that yields the following better upper bound on the price of hierarchy for radius and diameter.

► **Theorem 3.** For $\text{cost} \in \{\text{diam}, \text{rad}\}$ we have $\rho_{\text{cost}} \leq 3 + 2\sqrt{2} \approx 5.828$.

For the details we refer to the full version [4].

4 A Lower Bound on the Price of Hierarchy

The most challenging contributions of this article are matching lower bounds on the price of hierarchy for diameter, radius, and discrete radius.

► **Theorem 4.** *For $\text{cost} \in \{\text{diam}, \text{rad}\}$ we have $\rho_{\text{cost}} \geq 3 + 2\sqrt{2}$ and for $\text{cost} = \text{drad}$ we have $\rho_{\text{cost}} \geq 4$.*

There is already existing work in this area by Das and Kenyon-Mathieu [11] for the diameter and Großwendt [17] for the radius. Both show a lower bound of 2 for the respective objective. To improve upon these results we have to construct much more complex instances which differ significantly from those in [11, 17].

For every $\epsilon > 0$ we will construct a clustering instance $(\mathcal{X}, \mathcal{P}, d)$ such that for any hierarchical clustering $\mathcal{H} = (\mathcal{H}_{|\mathcal{P}|}, \dots, \mathcal{H}_1)$ of \mathcal{P} there is $1 \leq i \leq |\mathcal{P}|$ such that $\text{cost}(\mathcal{H}_i) \geq \alpha \cdot \text{cost}(\mathcal{O}_i)$, where \mathcal{O}_i is an optimal i -clustering of \mathcal{P} with respect to cost and $\alpha = (3 + 2\sqrt{2} - \epsilon)$ for $\text{cost} \in \{\text{diam}, \text{rad}\}$ and $\alpha = 4 - \epsilon$ for $\text{cost} = \text{drad}$.

The proof is divided in three parts. First we introduce the clustering instance $(\mathcal{X}, \mathcal{P}, d)$ and determine its optimal clusterings. In the second part we develop the notion of a *bad* cluster. We prove that any hierarchical clustering contains such bad clusters and develop a lower bound on their cost. In the third part we compare the lower bound to the cost of optimal clusterings and prove Theorem 4.

4.1 Definition of the Clustering Instance

For $n \in \mathbb{N}$ we denote by $[n]$ the set of numbers from 1 to n .

Let $k \in \mathbb{N}$ and $\Gamma = k + 1$. For $0 \leq \ell \leq k$ we define point sets \mathcal{Q}_ℓ and \mathcal{P}_ℓ recursively as follows:

1. For $\ell = 0$ let $\mathcal{P}_0 = \mathcal{Q}_0 = [1]$ and denote by N_0 the cardinality of \mathcal{P}_0 .
 2. For $\ell > 0$ let $\mathcal{Q}_\ell = [\Gamma \cdot N_{\ell-1}]^{N_{\ell-1}}$ and $\mathcal{P}_\ell = \prod_{i=0}^{\ell} \mathcal{Q}_i$. Furthermore set $N_\ell = |\mathcal{P}_\ell|$.
- Moreover let $\phi_\ell: \mathcal{P}_\ell \rightarrow [N_\ell]$ be a bijection for $0 \leq \ell \leq k$.

We refer to a point $X \in \mathcal{P}_k$ as a matrix with $k + 1$ rows and $N_{\ell-1}$ entries in the ℓ -th row. Thus we write

$$X = (x_{01} \mid \dots \mid x_{\ell 1}, \dots, x_{\ell N_{\ell-1}} \mid \dots \mid x_{k1}, \dots, x_{k N_{k-1}}).$$

Let $X_\ell = (x_{\ell 1}, \dots, x_{\ell N_{\ell-1}}) \in \mathcal{Q}_\ell$ for $0 \leq \ell \leq k$. For a shorter representation we can replace the ℓ -th row directly by X_ℓ and for $0 \leq i \leq j \leq k$ we can replace the i -th up to j -th row by $X_{[i:j]} = (X_i \mid \dots \mid X_j)$.

Let $X \in \mathcal{P}_k$ and $1 \leq \ell \leq k$. Notice that $X_{[0:\ell-1]} \in \mathcal{P}_{\ell-1}$ and let $m = \phi_{\ell-1}(X_{[0:\ell-1]})$, we define

$$A_\ell^X = \{(X_{[0:\ell-1]} \mid x_{\ell 1}, \dots, x_{\ell m-1}, \star, x_{\ell m+1}, \dots, x_{\ell N_{\ell-1}} \mid X_{[\ell+1:k]}) \mid \star \in [\Gamma \cdot N_{\ell-1}]\}.$$

Thus all coordinates of points in A_ℓ^X are fixed and agree with those of X except one which is variable. Here $X_{[0:\ell-1]}$ serves as prefix which indicates through $\phi_{\ell-1}$ which coordinate of X_ℓ can be changed.

We define $\mathcal{A}_\ell = \{A_\ell^X \mid X \in \mathcal{P}_k\}$ as the set containing all subsets of this form. It is clear that \mathcal{A}_ℓ is a partition of \mathcal{P}_k and that it contains only sets of size $\Gamma \cdot N_{\ell-1}$. Furthermore we set $\mathcal{A}_0 = \{\{X\} \mid X \in \mathcal{P}_k\}$.

► **Example 5.** If we perform the first three steps of the construction, we get $\mathcal{Q}_0 = [1]$, $\mathcal{Q}_1 = [\Gamma]$, $\mathcal{Q}_2 = [\Gamma^2]^\Gamma$ and

$$\mathcal{P}_1 = \{(1 \mid x_{11}) \mid x_{11} \in [\Gamma]\},$$

$$\mathcal{P}_2 = \{(1 \mid x_{11} \mid x_{21}, \dots, x_{2\Gamma}) \mid x_{11} \in [\Gamma], x_{2i} \in [\Gamma^2] \text{ for } 1 \leq i \leq \Gamma\}.$$

Since ϕ_0 is a map between two sets of cardinality one, this map is always unique. Now suppose that we picked ϕ_1 such that $\phi_1((1 \mid x_{11})) = x_{11}$ for all $(1 \mid x_{11}) \in \mathcal{P}_1$. Then the partition \mathcal{A}_1 consists of the sets

$$\{(1 \mid \star \mid x_{21}, \dots, x_{2\Gamma}) \mid \star \in [\Gamma]\}$$

with $x_{2i} \in [\Gamma^2]$ for all $1 \leq i \leq \Gamma$. The partition \mathcal{A}_2 consists of the sets

$$\{(1 \mid x_{11} \mid x_{21}, \dots, x_{2x_{11}-1}, \star, x_{2x_{11}+1}, \dots, x_{2\Gamma}) \mid \star \in [\Gamma^2]\}$$

with $x_{11} \in [\Gamma]$ and $x_{2i} \in [\Gamma^2]$ for all $1 \leq i \leq \Gamma$ with $i \neq x_{11}$. ┘

Now let $G = (V, E, w)$ denote the weighted hyper-graph with $V = \mathcal{P}_k$ and $E = \bigcup_{i=1}^k \mathcal{A}_i$. The weight of a hyper-edge $e \in E$ is set to ℓ iff $e \in \mathcal{A}_\ell$. For $0 \leq \ell \leq k$, the sub-graph $G_\ell = (V_\ell, E_\ell, w_\ell)$ is given by $V_\ell = \mathcal{P}_k$, $E_\ell = \bigcup_{i=0}^\ell \mathcal{A}_i$ and $w_\ell = w|_{E_\ell}$.

We extend G to a hyper-graph $H = (V', E', w')$ as follows. Let $V' = V \cup \bigcup_{i=0}^k \{v_A \mid A \in \mathcal{A}_i\}$ and $E' = E \cup \bigcup_{i=0}^k \{\{v, v_A\} \mid A \in \mathcal{A}_i, v \in A\}$. Thus H contains one vertex for every $A \in \bigcup_{i=0}^k \mathcal{A}_i$ and this vertex is connected by edges to every vertex $v \in A$. For $e \in E$ we set $w'(e) = w(e)$ and for $e = \{v, v_A\}$ for some $A \in \mathcal{A}_\ell$ and $v \in A$ we set $w'(e) = \ell/2$.

The clustering instance $(\mathcal{X}, \mathcal{P}, d)$ is given by $\mathcal{X} = V'$, $\mathcal{P} = V$, and d as the shortest path metric on H . Observe that the extension of G to H is only necessary for the lower bound for the radius but not for the diameter and the discrete radius. This is because the additional points $V' \setminus V$ do not belong to \mathcal{P} and are hence irrelevant for the clustering instance for the diameter and discrete radius. In the lower bound for the radius they will be used as centers, however.

► **Example 6.** For $k = 2$ we obtain $\Gamma = 3$ and $\mathcal{P} = \mathcal{P}_2$. Suppose that we again picked ϕ_1 such that $\phi_1((1 \mid x_{11})) = x_{11}$ for all $(1 \mid x_{11}) \in \mathcal{P}_1$. By the above definition the shortest path between the two points $X = (1 \mid 1 \mid 1, 1, 1)$, $Y = (1 \mid 1 \mid 2, 2, 1) \in \mathcal{P}_2$ in G is of the form

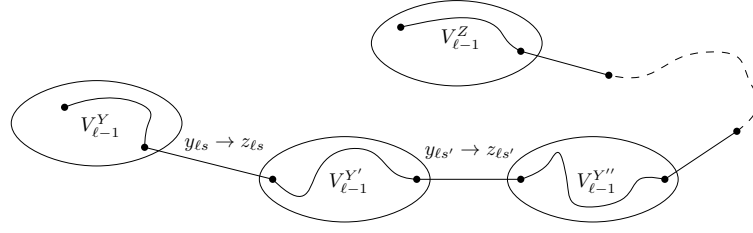
$$X = (1 \mid 1 \mid 1, 1, 1), (1 \mid 1 \mid 2, 1, 1), (1 \mid 2 \mid 2, 1, 1), (1 \mid 2 \mid 2, 2, 1), (1 \mid 1 \mid 2, 2, 1) = Y.$$

Thus the distance is given by $d(X, Y) = 2 + 1 + 2 + 1 = 6$. ┘

► **Lemma 7.** *Let $p, q \in V$, then $d(p, q)$ is the length of a shortest path between p and q in G .*

Proof. By definition $d(p, q)$ is the length of a shortest path between p and q in H . Suppose the shortest path contains a vertex v_A for some $A \in \bigcup_{i=0}^k \mathcal{A}_i$ with $v \in A$ as predecessor and $w \in A$ as ancestor. Since v and w are connected in H by the hyper-edge A we can delete v_A from the path and the length of the path does not change. The resulting path is also a path in G , so $d(p, q)$ is also the length of a shortest path between p and q in G . ◀

Next we state some structural properties of the graph G and the clustering instance $(\mathcal{X}, \mathcal{P}, d)$. To establish a lower bound on the approximation factor of a hierarchical clustering we first focus on the optimal clusterings of the instance $(\mathcal{X}, \mathcal{P}, d)$. One can already guess that \mathcal{A}_ℓ is an optimal clustering with $\frac{N_k}{\Gamma N_{\ell-1}}$ clusters with respect to $\text{cost} \in \{\text{diam}, \text{rad}, \text{drad}\}$ and we will prove this in this section. First we need the following statement about the connected components of G_ℓ .



■ **Figure 1** Here we see the construction of the path. It corresponds to changing the coordinates of Y successively until they match Z . We use an edge in \mathcal{A}_ℓ to change $y_{\ell s}$ to $z_{\ell s}$, next we change $y_{\ell s'}$ to $z_{\ell s'}$ and proceed like this until we obtain Z . The respective edges are then connected to a path from $V_{\ell-1}^X$ to $V_{\ell-1}^Z$.

► **Lemma 8.** *The vertex set of every connected component in G_ℓ has cardinality N_ℓ and is of the form $V_\ell^X = \{(X' \mid X) \mid X' \in \mathcal{P}_\ell\}$ for a given $X = (X_{\ell+1} \mid \dots \mid X_k) \in \prod_{i=\ell+1}^k \mathcal{Q}_i$.*

Proof. Notice that $|V_\ell^X| = N_\ell$ and that $\{V_\ell^X \mid X \in \prod_{i=\ell+1}^k \mathcal{Q}_i\}$ is a partition of V . Furthermore since $E_\ell = \bigcup_{i=0}^{\ell} \mathcal{A}_i$ any edge $e \in E_\ell$ is either completely contained in or disjoint to V_ℓ^X .

It is left to show that V_ℓ^X is connected. We prove this via induction over ℓ . For $\ell = 0$ this is clear because $|V_0^X| = 1$. For $\ell > 0$ let $Y = (Y_\ell \mid X), Z = (Z_\ell \mid X) \in \prod_{i=\ell}^k \mathcal{Q}_i$. By the induction hypothesis we know that the sets $V_{\ell-1}^Y, V_{\ell-1}^Z$ are connected. To prove that V_ℓ^X is connected it is sufficient to show that there is a path from a point in $V_{\ell-1}^Y$ to a point in $V_{\ell-1}^Z$. We show this claim by induction over the number m of coordinates in which Y and Z differ. For $m = 0$ there is nothing to show. If $m > 0$ pick $1 \leq s \leq N_{\ell-1}$ such that $y_{\ell s} \neq z_{\ell s}$ and let $P = \phi_{\ell-1}^{-1}(s) \in \prod_{i=0}^{\ell-1} \mathcal{Q}_i$. Consider the point $(P \mid Y_\ell \mid X)$ which is contained in $V_{\ell-1}^Y$. This point is also contained in the set

$$\{(P \mid y_{\ell 1}, \dots, y_{\ell s-1}, \star, y_{\ell s+1}, \dots, y_{\ell N_{\ell-1}} \mid X) \mid \star \in [\Gamma \cdot N_{\ell-1}]\} \in E_\ell.$$

Thus there is an edge in G_ℓ connecting a point in $V_{\ell-1}^Y$ to a point in $V_{\ell-1}^{Y'}$ with $Y' = (y_{\ell 1}, \dots, y_{\ell s-1}, z_{\ell s}, y_{\ell s+1}, \dots, y_{\ell N_{\ell-1}} \mid X)$. Now Y' and Z differ in $m - 1$ coordinates, thus there is a path between two points in $V_{\ell-1}^{Y'}$ and $V_{\ell-1}^Z$ by induction hypothesis. If we combine this with the induction hypothesis that $V_{\ell-1}^{Y'}$ is connected this yields the claim (see Figure 1 for an illustration). ◀

► **Lemma 9.** *Any clustering of $(\mathcal{X}, \mathcal{P}, d)$ with less than $\frac{N_k}{N_{\ell-1}}$ clusters costs at least ℓ if $\text{cost} \in \{\text{diam}, \text{drad}\}$ and $\ell/2$ if $\text{cost} = \text{rad}$.*

Proof. The shortest path in G between any two points which lie in different connected components of $G_{\ell-1}$ must contain an edge of weight $\geq \ell$. Thus any set of points $M \subset V$ which is disconnected in $G_{\ell-1}$ has diameter $\geq \ell$. Remember that the discrete radius of M is given by $\text{drad}(M) = \min_{c \in M} \max_{p \in M} d(p, c)$. For every possible choice of $c \in M$ there exists a point $p \in M$ which is not in the same connected component of $G_{\ell-1}$ as c , thus $d(c, p) \geq \ell$ and therefore $\text{drad}(M) \geq \ell$ and $\text{rad}(M) \geq \text{diam}(M)/2 \geq \ell/2$.

We conclude that if $\text{cost} \in \{\text{diam}, \text{drad}\}$ any cluster of cost smaller than ℓ is contained in one of the sets $V_{\ell-1}^X$ for some $X \in \prod_{i=\ell}^k \mathcal{Q}_i$ by Lemma 8 and any clustering with less than $|\prod_{i=\ell}^k \mathcal{Q}_i|$ clusters costs at least ℓ . By the same argument if $\text{cost} = \text{rad}$ any cluster of cost smaller than $\ell/2$ is contained in one of the sets $V_{\ell-1}^X$ for some $X \in \prod_{i=\ell}^k \mathcal{Q}_i$ by Lemma 8 and any clustering with less than $|\prod_{i=\ell}^k \mathcal{Q}_i|$ clusters costs at least $\ell/2$. Since

$$\left| \prod_{i=\ell}^k \mathcal{Q}_i \right| = \frac{\left| \prod_{i=0}^k \mathcal{Q}_i \right|}{\left| \prod_{i=0}^{\ell-1} \mathcal{Q}_i \right|} = \frac{N_k}{N_{\ell-1}}$$

this proves the lemma. \blacktriangleleft

► **Corollary 10.** For $1 \leq \ell \leq k$ and $\text{cost} \in \{\text{diam}, \text{rad}, \text{drad}\}$ the clustering \mathcal{A}_ℓ is an optimal $\frac{N_k}{\Gamma N_{\ell-1}}$ -clustering for the instance $(\mathcal{X}, \mathcal{P}, d)$. Furthermore $\text{diam}(\mathcal{A}_\ell) = \text{drad}(\mathcal{A}_\ell) = \ell$ and $\text{rad}(\mathcal{A}_\ell) = \ell/2$.

4.2 Characterization of Hierarchical Clusterings

Let from now on $\mathcal{H} = (\mathcal{H}_{N_k}, \dots, \mathcal{H}_1)$ denote a hierarchical clustering of $(\mathcal{X}, \mathcal{P}, d)$. We introduce the notion of *bad clusters* in $\mathcal{H}_{\frac{N_k}{\Gamma N_{\ell-1}}}$ which are clusters whose cost increases repeatedly, as we will see later. In this section we prove the existence of such clusters in \mathcal{H} and we give a lower bound on their cost.

► **Definition 11.** We call all clusters $C \in \mathcal{H}_{N_k}$ bad at time 0 and denote by $\text{Ker}_0(C) = C$ the kernel of C at time 0 and set $\text{Bad}(0) = \mathcal{H}_{N_k}$.

For $1 \leq \ell \leq k$ we say that a cluster $C \in \mathcal{H}_{\frac{N_k}{\Gamma N_{\ell-1}}}$ is anchored at $\ell \leq \ell' \leq k$ if the set

$$\bigcup_{D \in \text{Bad}(\ell-1): D \subset C} \text{Ker}_{\ell-1}(D) \text{ is}$$

1. connected in $G_{\ell'}$,
2. disconnected in $G_{\ell'-1}$.

We call C bad at time ℓ if C is anchored at some $\ell' \geq \ell$. We denote by $\text{Bad}(\ell) \subset \mathcal{H}_{\frac{N_k}{\Gamma N_{\ell-1}}}$ the set of all bad clusters at time ℓ . If C is bad we define the kernel of C as the union of all kernels of bad clusters at time $\ell - 1$ contained in C , i.e.,

$$\text{Ker}_\ell(C) = \bigcup_{D \in \text{Bad}(\ell-1): D \subset C} \text{Ker}_{\ell-1}(D).$$

All clusters in $\mathcal{H}_{\frac{N_k}{\Gamma N_{\ell-1}}} \setminus \text{Bad}(\ell)$ are called good.

The example in Figure 2 shows that a bad cluster at time ℓ can contain clusters which are good at time $\ell - 1$. However we are only interested in points that are contained exclusively in bad clusters at any time $t < \ell$. The set $\text{Ker}_\ell(C)$ contains exactly such points.

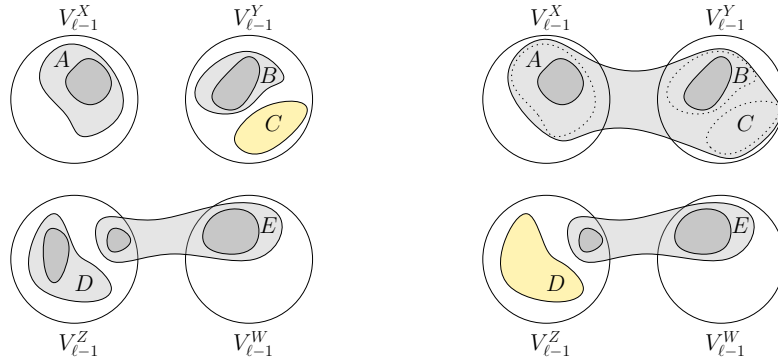
We will use two crucial properties to prove the final lower bound on the approximation factor of any hierarchical clustering \mathcal{H} of $(\mathcal{X}, \mathcal{P}, d)$. We first observe that bad clusters exist in \mathcal{H} for every time-step $1 \leq \ell \leq k$ and second that these clusters have a large cost compared to the optimal clustering.

► **Lemma 12.** For all $0 \leq \ell \leq k$ we have $\sum_{C \in \text{Bad}(\ell)} |\text{Ker}_\ell(C)| \geq \frac{\Gamma - \ell}{\Gamma} N_k$.

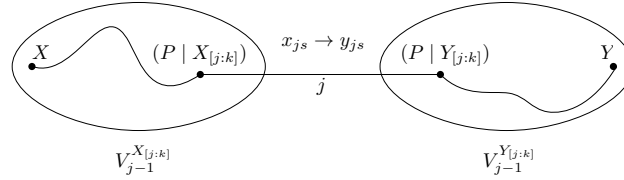
An immediate consequence of Lemma 12 is the existence of bad clusters at time ℓ for any $0 \leq \ell \leq k$. To prove that their (discrete) radius and diameter is indeed large we need a lower bound on the distance between two points $X, Y \in \mathcal{P}$ that lie in different connected components of G_{j-1} for some $1 \leq j \leq k$.

Suppose that the points X and Y only differ in one coordinate, i.e., there is a $1 \leq s \leq N_{j-1}$ such that $x_{js} \neq y_{js}$, while X and Y agree in all other coordinates. There is only one edge in G_j connecting $V_{j-1}^{X_{[j:k]}}$ with $V_{j-1}^{Y_{[j:k]}}$. Let $P = \phi_{j-1}^{-1}(s)$, then this edge connects the points

10:10 The Price of Hierarchical Clustering



■ **Figure 2** An illustration of the evolution of good and bad clusters: In the example, we see five clusters at time $\ell - 1$. The clusters A, B, D, E are assumed to be bad, with their kernels depicted in dark gray, while C is assumed to be a good cluster. At time ℓ , clusters A, B and C are merged. The resulting cluster is bad because the kernels of A and B lie in different connected components of $G_{\ell-1}$. Clusters D and E are still present at time ℓ , but now D is a good cluster since its kernel is completely contained in $V_{\ell-1}^Z$, while E is still bad, since its kernel is disconnected in $G_{\ell-1}$.



■ **Figure 3** A shortest path between X and Y . It consists of two shortest paths inside the connected components of G_{j-1} and the unique edge of weight j between these components.

$(P | X_{[j:k]})$ and $(P | Y_{[j:k]})$. If we connect X to $(P | X_{[j:k]})$ and $(P | Y_{[j:k]})$ to Y via a shortest path, this results in a path from X to Y , see Figure 3. We show that this path is indeed a shortest path between X and Y and generalize this to arbitrary X and Y which are disconnected in G_{j-1} .

► **Lemma 13.** *Let $X, Y \in \mathcal{P}$ be two points and suppose there is $1 \leq j \leq k$ and $1 \leq s \leq N_{j-1}$ such that $x_{js} \neq y_{js}$. Let $P = \phi_{j-1}^{-1}(s) \in \prod_{i=0}^{j-1} \mathcal{Q}_i$. Then*

$$d(X, Y) \geq d(X, (P | X_{[j:k]})) + j + d(Y, (P | Y_{[j:k]})).$$

We now define the so called *anchor set* $\text{Anc}_\ell(C)$ of a bad cluster C at time ℓ . If C is anchored at ℓ' then $\text{Anc}_\ell(C)$ is the union of ℓ' and the anchor set of some bad cluster $D \subset C$ at time $\ell - 1$. If we choose D appropriately the sum of anchors in $\text{Anc}_\ell(C)$ is a lower bound on the discrete radius of C , as we show later. It is clear that ℓ' itself is a lower bound on the discrete radius since $\text{Ker}_\ell(C)$ is disconnected in $G_{\ell'-1}$ by definition. If we additionally assume that the discrete radius of D is large, e.g., lower bounded by the sum of anchors in $\text{Anc}_{\ell-1}(D)$, then it is reasonable to assume that the discrete radius of C is lower bounded by some function in ℓ' and the sum of anchors in $\text{Anc}_{\ell-1}(D)$. First we give a formal definition of $\text{Anc}_\ell(C)$ and how to choose D .

► **Definition 14.** *Let $1 \leq \ell \leq k$ and C be a bad cluster at time ℓ which is anchored at $\ell' \geq \ell$. If $\ell = 1$ we define the anchor set of C as $\text{Anc}_1(C) = \{\ell'\}$ and set $\text{prev}(C) = \{X\}$ for some $X \in C$.*

For $\ell > 1$ we distinguish two cases.

Case 1: C contains a bad cluster D which is bad at time $\ell - 1$ and anchored at ℓ' . We then set $\text{Anc}_\ell(C) = \text{Anc}_{\ell-1}(D)$ and $\text{prev}(C) = D$.

Case 2: C does not contain such a cluster. Then let $D \subset C$ be a bad cluster at time $\ell - 1$ minimizing

$$\sum_{a \in \text{Anc}_{\ell-1}(D)} a$$

among all clusters $D' \in \text{Bad}(\ell - 1)$ with $D' \subset C$. We set $\text{Anc}_\ell(C) = \text{Anc}_{\ell-1}(D) \cup \{\ell'\}$ and $\text{prev}(C) = D$.

Observe that in Case 2 of the previous definition, the bad cluster D must be anchored at some $\ell_D < \ell'$.

With the help of Lemma 13 we are able to show how the discrete radius and diameter of a bad cluster, depends on the sum of anchors.

► **Lemma 15.** *Let $1 \leq \ell \leq k$ and C be a bad cluster at time ℓ anchored at ℓ' . Then for any point $Z \in \mathcal{P}$ there is $X \in \text{Ker}_\ell(C)$ such that*

$$d(Z, X) \geq \sum_{a \in \text{Anc}_\ell(C)} a.$$

► **Lemma 16.** *Let $1 \leq \ell \leq k$ and C be a bad cluster at time ℓ anchored at ℓ' . Then there are two points $X, Y \in \text{Ker}_\ell(C)$ such that*

$$d(X, Y) \geq \ell' + 2 \sum_{a \in \text{Anc}_\ell(C) \setminus \{\ell'\}} a.$$

4.3 Comparison to Optimal Clusterings

Our initial motivation was to construct an instance where any hierarchical clustering has a high approximation ratio. If we consider an arbitrary time $1 \leq \ell \leq k$ then the hierarchical clustering \mathcal{H} on $(\mathcal{X}, \mathcal{P}, d)$ may be even optimal at time ℓ . Thus the bounds which we develop in Lemma 15 and Lemma 16 on the discrete radius and diameter of bad clusters are useless without linking the cost of a bad cluster at time ℓ to the cost of bad clusters at other time steps. Therefore we construct a sequence of clusters $C_1 \subset C_2 \dots \subset C_k$ where C_i is a bad cluster at time i such that $\text{Anc}_1(C_1) \subset \text{Anc}_2(C_2) \subset \dots \subset \text{Anc}_k(C_k)$. We then show with the help of Lemma 15 and Lemma 16 that at least one of these clusters has a high discrete radius and diameter compared to the optimal cost.

► **Lemma 17.** *Let C_k be a bad cluster at time k . For $1 \leq i \leq k - 1$ we define $C_i = \text{prev}(C_{i+1})$. For all $1 \leq i \leq k - 1$ cluster C_i is bad at time i and one of the following two cases occurs:*

1. $\text{Anc}_i(C_i) = \text{Anc}_{i+1}(C_{i+1})$,
2. $\text{Anc}_{i+1}(C_{i+1}) \setminus \{\ell\} = \text{Anc}_i(C_i)$, where $\ell = \max \text{Anc}_{i+1}(C_{i+1})$.

Proof. For $i = k$ cluster C_k is bad at time k by assumption. If C_{i+1} is a bad cluster at time $i + 1$ then $C_i = \text{prev}(C_{i+1})$ is a bad cluster at time i , by definition of prev .

Let C_i be anchored at $\ell' \geq i$ and C_{i+1} be anchored at $\ell \geq i + 1$. Since C_i is a bad cluster at time i with $C_i \subset C_{i+1}$ we have by definition of $\text{Ker}_{i+1}(C_{i+1})$ that $\text{Ker}_i(C_i) \subset \text{Ker}_{i+1}(C_{i+1})$ and thus $\ell' \leq \ell$. If $\ell' = \ell$ we obtain by Definition 14, that $\text{Anc}_i(C_i) = \text{Anc}_{i+1}(C_{i+1})$, so the lemma holds in this case.

If $\ell' < \ell$ we know by Definition 14 that $\text{Anc}_i(C_i) = \text{Anc}_{i+1}(C_{i+1}) \setminus \{\ell\}$. So the lemma also holds in this case. ◀

10:12 The Price of Hierarchical Clustering

► **Corollary 18.** *Let C_k be a bad cluster at time k . For $1 \leq i \leq k-1$ we define $C_i = \text{prev}(C_{i+1})$. Let $\text{Anc}_k(C_k) = \{\ell_1, \dots, \ell_s\}$ such that $\ell_{t-1} < \ell_t$ for all $2 \leq t \leq s$ and let $\ell_0 = 0$. Then for any $1 \leq t \leq s$ and for any i with $\ell_{t-1} < i \leq \ell_t$, we have $\{\ell_1, \dots, \ell_t\} \subset \text{Anc}_i(C_i)$.*

Proof. We prove this via induction over i , starting from $i = k$ in decreasing order. There is nothing to show for $i = k$. For $i < k$ we distinguish two cases. If $\text{Anc}(C_i) = \text{Anc}_{i+1}(C_{i+1})$, the lemma follows from the induction hypothesis.

Otherwise remember that $\text{Anc}_i(C_i) \subset \text{Anc}_k(C_k)$ and $\ell_{t-1} < i$. Since $\max \text{Anc}_i(C_i) \geq i$ we obtain that $\max \text{Anc}_i(C_i) \in \{\ell_t, \dots, \ell_s\}$ and therefore $\ell_t \leq \max \text{Anc}_i(C_i)$.

By Lemma 17 we know that $\text{Anc}_i(C_i) = \text{Anc}_{i+1}(C_{i+1}) \setminus \{\ell\}$, where $\ell = \max \text{Anc}_{i+1}(C_{i+1})$. Thus $\ell_t \leq \max \text{Anc}_i(C_i) < \max \text{Anc}_{i+1}(C_{i+1}) = \ell$ and by induction hypothesis we obtain

$$\{\ell_1, \dots, \ell_t\} \subset \text{Anc}_{i+1}(C_{i+1}) \setminus \{\ell\} = \text{Anc}_i(C_i). \quad \blacktriangleleft$$

Before we are able to prove the theorem we need some final lemma.

► **Lemma 19.** *For every $\epsilon > 0$ there exists $k \in \mathbb{N}$ such that for every $s \in \mathbb{N}$ any sequence of $s+1$ numbers $(\ell_0, \dots, \ell_s) \in \mathbb{R}_{\geq 0}^{s+1}$ with $\ell_0 = 0$ and $\ell_s = k$ satisfies the following.*

1. *There exists $1 \leq t \leq s$ such that for $\alpha_1 = 4 - \epsilon$ and $\Delta_1 = 1$ we have*

$$\frac{\ell_t + \Delta_1 \sum_{i=0}^{t-1} \ell_i}{\ell_{t-1} + 1} > \alpha_1.$$

2. *There exists $1 \leq t \leq s$ such that for $\alpha_2 = 3 + 2\sqrt{2} - \epsilon$ and $\Delta_2 = 2$ we have*

$$\frac{\ell_t + \Delta_2 \sum_{i=0}^{t-1} \ell_i}{\ell_{t-1} + 1} > \alpha_2.$$

► **Theorem 4.** *For $\text{cost} \in \{\text{diam}, \text{rad}\}$ we have $\rho_{\text{cost}} \geq 3 + 2\sqrt{2}$ and for $\text{cost} = \text{drad}$ we have $\rho_{\text{cost}} \geq 4$.*

Proof. Let $\epsilon > 0$ and k be the respective number from Lemma 19. We claim that the approximation factor of any hierarchical clustering $\mathcal{H} = (\mathcal{H}_{N_k}, \dots, \mathcal{H}_1)$ on the instance $(\mathcal{X}, \mathcal{P}, d)$ is larger than $3 + 2\sqrt{2} - \epsilon$ if $\text{cost} \in \{\text{diam}, \text{rad}\}$ and larger than $4 - \epsilon$ if $\text{cost} = \text{drad}$. First we use Lemma 12 to observe that there is a cluster $C_k \in \mathcal{H}_{\frac{N_k}{\Gamma N_{k-1}}}$ that is bad at time k . For $1 \leq i \leq k-1$ we define $C_i = \text{prev}(C_{i+1})$. Let $\text{Anc}_k(C_k) = \{\ell_1, \dots, \ell_s\}$ with $\ell_{t-1} < \ell_t$ for $2 \leq t \leq s$ and let $\ell_0 = 0$. We know by Corollary 18, that for any $1 \leq t \leq s$ and for $i = \ell_{t-1} + 1$ we have $\{\ell_1, \dots, \ell_t\} \subset \text{Anc}_i(C_i)$. Let $\ell' = \max \text{Anc}_i(C_i)$, we obtain by Lemma 16 and Lemma 15 that

$$\begin{aligned} \text{diam}(C_i) &\geq \ell' + 2 \sum_{a \in \text{Anc}_i(C_i) \setminus \{\ell'\}} a \geq \ell_t + 2 \sum_{u=1}^{t-1} \ell_u, \\ \text{rad}(C_i) &\geq \frac{\text{diam}(C_i)}{2} \geq \frac{\ell_t + 2 \sum_{u=1}^{t-1} \ell_u}{2}, \\ \text{drad}(C_i) &\geq \sum_{a \in \text{Anc}_i(C_i)} a \geq \sum_{u=1}^t \ell_u. \end{aligned}$$

Remember that by Corollary 10 \mathcal{A}_i is an optimal $\frac{N_k}{\Gamma N_{i-1}}$ -clustering with $\text{cost}(\mathcal{A}_i) = i$ if $\text{cost} \in \{\text{diam}, \text{drad}\}$ and $\text{cost}(\mathcal{A}_i) = i/2$ if $\text{cost} = \text{rad}$. We obtain

$$\frac{\text{rad}(C_i)}{\text{rad}(\mathcal{A}_i)} = \frac{2\text{rad}(C_i)}{2\text{rad}(\mathcal{A}_i)} \geq \frac{\text{diam}(C_i)}{\text{diam}(\mathcal{A}_i)} \geq \frac{\ell_t + 2\sum_{u=1}^{t-1} \ell_u}{\ell_{t-1} + 1}$$

$$\frac{\text{drad}(C_i)}{\text{drad}(\mathcal{A}_i)} \geq \frac{\sum_{u=1}^t \ell_u}{\ell_{t-1} + 1}$$

which are lower bounds on the approximation factor of \mathcal{H} .

We apply Lemma 19 on (ℓ_0, \dots, ℓ_s) to observe that there is $1 \leq t' \leq s$ such that

$$\frac{\ell_{t'} + 2\sum_{u=1}^{t'-1} \ell_u}{\ell_{t'-1} + 1} > 3 + 2\sqrt{2} - \epsilon$$

and an $1 \leq t'' \leq s$ such that

$$\frac{\sum_{u=1}^{t''} \ell_u}{\ell_{t''-1} + 1} > 4 - \epsilon.$$

This proves the theorem. ◀

5 Conclusions and Open Problems

We have proved tight bounds for the price of hierarchy with respect to the diameter and (discrete) radius. It would be interesting to also obtain a better understanding of the price of hierarchy for other important objective functions like k -median and k -means. The best known upper bound is 16 for k -median [10] and 32 for k -means [17] but no non-trivial lower bounds are known. Closing this gap also for these objectives is a challenging problem for further research.

Another natural question is which approximation factors can be achieved by polynomial-time algorithms. The algorithm we used in this article to prove the upper bounds is not a polynomial-time algorithm because it assumes that for each level k an optimal k -clustering is given. The approximation factors worsen if only approximately optimal clusterings are used instead. It is known that 8-approximate hierarchical clusterings can be computed efficiently with respect to the diameter and (discrete) radius [13]. It is not clear whether or not it is NP-hard to obtain better hierarchical clusterings. The only NP-hardness results come from the problems with given k . Since computing a $(2 - \epsilon)$ -approximation for k -clustering with respect to the diameter and (discrete) radius is NP-hard, this is also true for the hierarchical versions. However, this is obsolete due to our lower bound, which shows that in general there does not even exist a $(2 - \epsilon)$ -approximate hierarchical clustering.

References

- 1 Marcel R. Ackermann, Johannes Blömer, Daniel Kuntze, and Christian Sohler. Analysis of agglomerative clustering. *Algorithmica*, 69(1):184–215, 2014. doi:10.1007/s00453-012-9717-4.
- 2 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k -means and Euclidean k -median by primal-dual algorithms. In *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72, 2017. doi:10.1109/FOCS.2017.15.
- 3 Anna Arutyunova, Anna Großwendt, Heiko Röglin, Melanie Schmidt, and Julian Wargalla. Upper and lower bounds for complete linkage in general metric spaces. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 18:1–18:22, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.18.

- 4 Anna Arutyunova and Heiko Röglin. The price of hierarchical clustering. *CoRR*, abs/2205.01417, 2022. doi:10.48550/arXiv.2205.01417.
- 5 Felix Bock. Hierarchy cost of hierarchical clusterings. *Journal of Combinatorial Optimization*, 2022. doi:10.1007/s10878-022-00851-4.
- 6 Jaroslaw Byrka, Thomas W. Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k -median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2):23:1–23:31, 2017. doi:10.1145/2981561.
- 7 Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 841–854, 2017. doi:10.1137/1.9781611974782.53.
- 8 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.
- 9 Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. In *Proc. of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 378–397, 2018. doi:10.1137/1.9781611975031.26.
- 10 Wenqiang Dai. A 16-competitive algorithm for hierarchical median problem. *SCIENCE CHINA Information Sciences*, 57(3):1–7, 2014. doi:10.1007/s11432-014-5065-0.
- 11 Aparna Das and Claire Kenyon-Mathieu. On hierarchical diameter-clustering and the supplier problem. *Theory Comput. Syst.*, 45(3):497–511, 2009. doi:10.1007/s00224-009-9186-6.
- 12 Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proc. of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 118–127, 2016. doi:10.1145/2897518.2897527.
- 13 Sanjoy Dasgupta and Philip M. Long. Performance guarantees for hierarchical clustering. *Journal of Computer and System Sciences*, 70(4):555–569, 2005. doi:10.1016/j.jcss.2004.10.006.
- 14 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.
- 15 Anna Großwendt and Heiko Röglin. Improved analysis of complete-linkage clustering. *Algorithmica*, 78(4):1131–1150, 2017. doi:10.1007/s00453-017-0284-6.
- 16 Anna Großwendt, Heiko Röglin, and Melanie Schmidt. Analysis of ward’s method. In *Proc. of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2939–2957, 2019. doi:10.1137/1.9781611975482.182.
- 17 Anna-Klara Großwendt. *Theoretical Analysis of Hierarchical Clustering and the Shadow Vertex Algorithm*. PhD thesis, University of Bonn, 2020. URL: <http://hdl.handle.net/20.500.11811/8348>.
- 18 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986. doi:10.1145/5925.5933.
- 19 Guolong Lin, Chandrashekar Nagarajan, Rajmohan Rajaraman, and David P. Williamson. A general approach for incremental approximation and hierarchical clustering. *SIAM Journal on Computing*, 39(8):3633–3669, 2010. doi:10.1137/070698257.
- 20 Sakib A. Mondal. An improved approximation algorithm for hierarchical clustering. *Pattern Recognit. Lett.*, 104:23–28, 2018. doi:10.1016/j.patrec.2018.01.015.
- 21 C. Greg Plaxton. Approximation algorithms for hierarchical location problems. *Journal of Computer and System Sciences*, 72(3):425–443, 2006. doi:10.1016/j.jcss.2005.09.004.
- 22 Yuyan Wang and Benjamin Moseley. An objective for hierarchical clustering in euclidean space and its connection to bisecting k -means. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):6307–6314, 2020. doi:10.1609/aaai.v34i04.6099.
- 23 Joe H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. doi:10.1080/01621459.1963.10500845.

Bounding and Computing Obstacle Numbers of Graphs

Martin Balko  

Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Robert Ganian  

Algorithms and Complexity Group,
TU Wien, Austria

Michael Hoffmann  

Department of Computer Science,
ETH Zürich, Switzerland

Alexander Wolff 

Institut für Informatik,
Universität Würzburg, Germany

Steven Chaplick 

Maastricht University, The Netherlands

Siddharth Gupta  

Department of Computer Science,
University of Warwick, Coventry, UK

Pavel Valtr 

Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

Abstract

An *obstacle representation* of a graph G consists of a set of pairwise disjoint simply-connected closed regions and a one-to-one mapping of the vertices of G to points such that two vertices are adjacent in G if and only if the line segment connecting the two corresponding points does not intersect any obstacle. The *obstacle number* of a graph is the smallest number of obstacles in an obstacle representation of the graph in the plane such that all obstacles are simple polygons.

It is known that the obstacle number of each n -vertex graph is $O(n \log n)$ [Balko, Cibulka, and Valtr, 2018] and that there are n -vertex graphs whose obstacle number is $\Omega(n/(\log \log n)^2)$ [Dujmović and Morin, 2015]. We improve this lower bound to $\Omega(n/\log \log n)$ for simple polygons and to $\Omega(n)$ for convex polygons. To obtain these stronger bounds, we improve known estimates on the number of n -vertex graphs with bounded obstacle number, solving a conjecture by Dujmović and Morin. We also show that if the drawing of some n -vertex graph is given as part of the input, then for some drawings $\Omega(n^2)$ obstacles are required to turn them into an obstacle representation of the graph. Our bounds are asymptotically tight in several instances.

We complement these combinatorial bounds by two complexity results. First, we show that computing the obstacle number of a graph G is fixed-parameter tractable in the vertex cover number of G . Second, we show that, given a graph G and a simple polygon P , it is NP-hard to decide whether G admits an obstacle representation using P as the only obstacle.

2012 ACM Subject Classification Theory of computation → Computational geometry; Human-centered computing → Graph drawings

Keywords and phrases Obstacle representation, Obstacle number, Visibility, NP-hardness, FPT

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.11

Related Version *Full Version*: <http://arxiv.org/abs/2206.15414> [3]

Funding *Martin Balko*: Grant no. 21-32817S of the Czech Science Foundation (GAČR) and support by the Center for Foundations of Modern Computer Science (Charles University project UNCE/S-CI/004) and by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement no. 810115).

Robert Ganian: Project no. Y1329 of the Austrian Science Fund (FWF).

Siddharth Gupta: Engineering and Physical Sciences Research Council (EPSRC) grant EP/V007793/1.

Michael Hoffmann: Swiss National Science Foundation within the collaborative DACH project *Arrangements and Drawings* as SNSF Project 200021E-171681.

Pavel Valtr: Grant no. 21-32817S of the Czech Science Foundation (GAČR).

Alexander Wolff: DFG–GAČR project Wo 754/11-1.



© Martin Balko, Steven Chaplick, Robert Ganian, Siddharth Gupta, Michael Hoffmann, Pavel Valtr, and Alexander Wolff;

licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 11; pp. 11:1–11:13



Leibniz International Proceedings in Informatics

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

Acknowledgements This research was initiated at the Bertinoro Workshop on Graph Drawing 2022 (BWGD'22).

1 Introduction

An *obstacle* is a simple polygon in the plane. For a set S of points in the plane and a set \mathcal{O} of obstacles, the *visibility graph* of S with respect to \mathcal{O} is the graph with vertex set S where two vertices s and t are adjacent if and only if the line segment \overline{st} does not intersect any obstacle in \mathcal{O} . For convenience, we identify vertices with the points representing them and edges with the corresponding line segments. An *obstacle representation* of a graph G consists of a set \mathcal{O} of pairwise disjoint obstacles and a bijective mapping between the vertex set of G and a point set $S \subseteq \mathbb{R}^2 \setminus \bigcup \mathcal{O}$ such that the visibility graph of S with respect to \mathcal{O} is isomorphic to G . We consider finite point sets and finite collections of obstacles. All obstacles are closed. For simplicity, we consider point sets to be in *general position*, that is, no three points lie on a common line.

Given a straight-line drawing D of a graph G , we define the *obstacle number* of D , $\text{obs}(D)$, to be the smallest number of obstacles that are needed in order to turn D into an obstacle representation of G . Since every *non-edge* of G (that is, every pair of non-adjacent vertices) must be blocked by an obstacle and no edge of G may intersect an obstacle, $\text{obs}(D)$ is the cardinality of the smallest set of faces of D whose union is intersected by every non-edge of G . For a graph G , the *obstacle number* of G , $\text{obs}(G)$, is the smallest value of $\text{obs}(D)$, where D ranges over all straight-line drawings of G . For a positive integer n , let $\text{obs}(n)$ be the maximum value of $\text{obs}(G)$, where G is any n -vertex graph. The *convex obstacle number* $\text{obs}_c(\cdot)$ is defined analogously, except that here all obstacles are convex polygons.

For positive integers h and n , let $f(h, n)$ be the number of graphs on n vertices that have obstacle number at most h . Similarly, we denote the number of graphs on n vertices that have convex obstacle number at most h by $f_c(h, n)$.

Alpert, Koch, and Laison [2] introduced the obstacle number and the convex obstacle number. Using Ramsey theory, they proved that for every positive integer k , there is a (huge) complete k -partite graph with convex obstacle number k . They also showed that $\text{obs}(n) \in \Omega(\sqrt{\log n / \log \log n})$. This lower bound was subsequently improved to $\Omega(n / \log n)$ by Mukkamala, Pach, and Pálvölgyi [18] and to $\Omega(n / (\log \log n)^2)$ by Dujmović and Morin [8], who conjectured the following.

► **Conjecture 1** ([8]). *For all positive integers n and h , we have $f(h, n) \in 2^{g(n) \cdot o(h)}$, where $g(n) \in O(n \log^2 n)$.*

On the other hand, we trivially have $\text{obs}(n) \leq \text{obs}_c(n) \leq \binom{n}{2}$ as one can block each non-edge of an n -vertex graph with a single obstacle. Balko, Cibulka, and Valtr [4] improved this upper bound to $\text{obs}(n) \leq \text{obs}_c(n) \leq n \lceil \log n \rceil - n + 1$, refuting a conjecture by Mukkamala, Pach, and Pálvölgyi [18] stating that $\text{obs}(n)$ is around n^2 . For every graph G with chromatic number χ , Balko, Cibulka, and Valtr [4] showed that $\text{obs}_c(G) \leq (n - 1)(\lceil \log \chi \rceil + 1)$, which is in $O(n)$ if the chromatic number is bounded by a constant.

Alpert, Koch, and Laison [2] differentiated between an *outside obstacle*, which lies in (or simply is) the outer face of the visibility drawing, and *inside obstacles*, which lie in (or are) inner faces of the drawing. They proved that every outerplanar graph has an outside-obstacle representation, that is, a representation with a single outside obstacle. Later, Chaplick, Lipp, Park, and Wolff [6] showed that the class of graphs that admit a representation with a single inside obstacle is incomparable with the class of graphs that have an outside-obstacle representation. They found the smallest graph with obstacle number 2; it has eight vertices

and is co-bipartite. They also showed that the following *sandwich version* of the outside-obstacle representation problem is NP-hard: Given two graphs G and H with $V(G) = V(H)$ and $E(G) \subseteq E(H)$, is there a graph K with $V(K) = V(G)$ and $E(G) \subseteq E(K) \subseteq E(H)$ that admits an outside-obstacle representation? Analogous hardness results hold with respect to inside and general obstacles. Every partial 2-tree has an outside-obstacle representation [10]. For (partial) outerpaths, cactus graphs, and grids, it is possible to construct outside-obstacle representations where the vertices are those of a regular polygon [10].

For planar graphs, Johnson and Sariöz [16] investigated a variant of the problem where the visibility graph G is required to be plane and a plane drawing D of G is given. They showed that computing $\text{obs}(D)$ is NP-hard (by reduction *from* planar vertex cover) and that there is a solution-value-preserving reduction *to* maximum-degree-3 planar vertex cover. For computing $\text{obs}(D)$, this reduction yields a polynomial-time approximation scheme and a fixed-parameter algorithm with respect to $\text{obs}(D)$. Gimbel, de Mendez, and Valtr [13] showed that, for some planar graphs, there is a large discrepancy between this planar setting and the usual obstacle number.

A related problem deals with *point visibility graphs*, where the points are not only the vertices of the graph but also the obstacles. Recognizing point visibility graphs is contained in $\exists\mathbb{R}$ [12] and is $\exists\mathbb{R}$ -hard [5], and thus $\exists\mathbb{R}$ -complete.

Our Contribution. Our results span three areas. First, we improve existing bounds on the (convex and general) obstacle number (see Section 2), on the functions $f(n, h)$ and $f_c(n, h)$ (see Section 3), and on the obstacle number of drawings (see Section 4). Second, we provide an algorithmic result: computing the obstacle number of a given graph is fixed-parameter tractable with respect to the vertex cover number of the graph (see Section 5). Third, we show that, given a graph G and a simple polygon P , it is NP-hard to decide whether G admits an obstacle representation using P as the only obstacle (see Section 6). The omitted proofs are available in the full version of this paper [3]. We now describe our results in more detail.

First, we prove the currently strongest lower bound on the obstacle number of n -vertex graphs, improving the estimate of $\text{obs}(n) \in \Omega(n/(\log \log n)^2)$ by Dujmović and Morin [8].

► **Theorem 2.** *There is a constant $\beta > 0$ such that, for every $n \in \mathbb{N}$, there exists a graph on n vertices with obstacle number at least $\beta n / \log \log n$, that is, $\text{obs}(n) \in \Omega(n / \log \log n)$.*

This lower bound is quite close to the currently best upper bound $\text{obs}(n) \in O(n \log n)$ by Balko, Cibulka, and Valtr [4]. In fact, Alpert, Koch, and Laison [2] asked whether the obstacle number of any graph with n vertices is bounded from above by a linear function of n . This is supported by a result of Balko, Cibulka, and Valtr [4] who proved $\text{obs}(G) \leq \text{obs}_c(G) \in O(n)$ for every graph G with n vertices and with constant chromatic number. We remark that we are not aware of any argument that would give a strengthening of Theorem 2 to graphs with constant chromatic number.

Next, we show that a linear lower bound holds for convex obstacles.

► **Theorem 3.** *There is a constant $\gamma > 0$ such that, for every $n \in \mathbb{N}$, there exists a (bipartite) graph on n vertices with convex obstacle number at least γn , that is, $\text{obs}_c(n) \in \Omega(n)$.*

The previously best known bound on the convex obstacle number was $\text{obs}_c(G) \in \Omega(n/(\log \log n)^2)$ [8]. Recall that the upper bound proved by Balko, Cibulka, and Valtr [4] actually holds for the convex obstacle number as well and gives $\text{obs}_c(n) \in O(n \log n)$. Furthermore, the linear lower bound on $\text{obs}_c(n)$ from Theorem 3 works for n -vertex graphs with bounded chromatic number, asymptotically matching the linear upper bound on the obstacle number of such graphs proved by Balko, Cibulka, and Valtr [4]; see the remark in Section 2.

11:4 Bounding and Computing Obstacle Numbers of Graphs

The proofs of Theorems 2 and 3 are both based on counting arguments that use the upper bounds on $f(h, n)$ and $f_c(h, n)$. To obtain the stronger estimates on $\text{obs}(n)$ and $\text{obs}_c(n)$, we improve the currently best bounds $f(h, n) \in 2^{O(hn \log^2 n)}$ and $f_c(h, n) \in 2^{O(hn \log n)}$ by Muckkamala, Pach, and Pálvölgyi [18] as follows.

► **Theorem 4.** *For all positive integers h and n , we have $f(h, n) \in 2^{O(hn \log n)}$.*

► **Theorem 5.** *For all positive integers h and n , we have $f_c(h, n) \in 2^{O(n(h + \log n))}$.*

The upper bound in Theorem 5 is asymptotically tight for $h < n$ as Balko, Cibulka, and Valtr [4] proved that for every pair of integers n and h satisfying $0 < h < n$ we have $f_c(h, n) \in 2^{\Omega(hn)}$. Their result is stated for $f(h, n)$, but it is proved using convex obstacles only. This matches our bound for $h \in \Omega(\log n)$. Moreover, they showed that $f_c(1, n) \in 2^{\Omega(n \log n)}$, which again matches the bound in Theorem 5.

Since trivially $h \leq \binom{n}{2} \leq n^2$, we get $\log h \leq 2 \log n$, and thus the bound in Theorem 4 can be rewritten as $f(h, n) \leq 2^{g(n) \cdot (h/\log n)} \leq 2^{g(n) \cdot (2h/\log h)} \in 2^{g(n) \cdot o(h)}$, where $g(n) \in O(n \log^2 n)$. Therefore, we get the following corollary, confirming Conjecture 1.

► **Corollary 6.** *For all positive integers h and n , we have $f(h, n) \in 2^{g(n) \cdot (2h/\log h)}$, where $g(n) \in O(n \log^2 n)$.*

It is natural to ask about estimates on obstacle numbers of fixed drawings, that is, considering the problem of estimating $\text{obs}(D)$. The parameter $\text{obs}(D)$ has been considered in the literature; e.g., by Johnson and Sariöz [16]. Here, we provide a quadratic lower bound on $\max_D \text{obs}(D)$ where the maximum ranges over all drawings of graphs on n vertices.

► **Theorem 7.** *There is a constant $\delta > 0$ such that, for every n , there exists a graph G on n vertices and a drawing D of G such that $\text{obs}(D) \geq \delta \cdot n^2$.*

The bound from Theorem 7 is asymptotically tight as we trivially have $\text{obs}(D) \leq \text{obs}_c(D) \leq \binom{n}{2}$ for every drawing D of an n -vertex graph. This also asymptotically settles the problem of estimating the convex obstacle number of drawings.

Next, we turn our attention to algorithms for computing the obstacle number. We establish fixed-parameter tractability for the problem when parameterized by the size of a minimum vertex cover of the input graph G , called the *vertex cover number* of G .

► **Theorem 8.** *Given a graph G and an integer h , the problem of recognizing whether G admits an obstacle representation with h obstacles is fixed-parameter tractable parameterized by the vertex cover number of G .*

The proof of Theorem 8 is surprisingly non-trivial. On a high level, it uses Ramsey techniques to identify, in a sufficiently large graph G , a set of vertices outside a minimum vertex cover which not only have the same neighborhood, but also have certain geometric properties in a hypothetical solution S . We then use a combination of topological and combinatorial arguments to show that S can be adapted to an equivalent solution for a graph G' whose size is bounded by the vertex cover number of G – i.e., a kernel [7].

While the complexity of deciding whether a given graph has obstacle number 1 is still open, the sandwich version of the problem [6] and the version for planar visibility drawings [16] have been shown NP-hard. We conclude with a simple new NP-hardness result.

► **Theorem 9.** *Given a graph G and a simple polygon P , it is NP-hard to decide whether G admits an obstacle representation using P as (outside-) obstacle.*

2 Improved Lower Bounds on Obstacle Numbers

We start with the estimate $\text{obs}(n) \in \Omega(n/\log \log n)$ from Theorem 2. The proof is based on Theorem 4 (cf. Section 3) and the following result by Dujmović and Morin [8], which shows that an upper bound for $f(h, n)$ translates into a lower bound for $\text{obs}(n)$.

► **Theorem 10** ([8]). *For every $k \in \mathbb{N}$, let $H(k) = \max\{h : f(h, k) \leq 2^{k^2/4}\}$. Then, for any constant $c > 0$, there exist n -vertex graphs with obstacle number at least $\Omega\left(\frac{nH(c \log n)}{c \log n}\right)$.*

Proof of Theorem 2. By Theorem 4, we have $f(h, k) \in 2^{O(hk \log k)}$ for every positive integer k and thus $H(k) \in \Omega(k/\log k)$. Plugging this estimate with $k \in \Omega(\log n)$ into Theorem 10, we get the lower bound $\text{obs}(n) \in \Omega(n/\log \log n)$. ◀

It remains to prove the lower bound $\text{obs}_c(n) \in \Omega(n)$ from Theorem 3.

Proof of Theorem 3. By Theorem 5, we have $f_c(h, n) \leq 2^{\beta n(h+\log n)}$ for some constant $\beta > 0$. Since the number of graphs with n vertices is $2^{\binom{n}{2}}$, we see that if $2^{\binom{n}{2}} > 2^{\beta n(h+\log n)}$, then there is an n -vertex graph with convex obstacle number larger than h . The inequality $\binom{n}{2} > \beta n(h + \log n)$ is satisfied for $h < \binom{n}{2}/(\beta n) - \log n \in \Theta(n)$. Thus, $\text{obs}_c(n) \in \Omega(n)$. ◀

► **Remark.** Observe that the proof of Theorem 3 also works if we restrict ourselves to bipartite graphs, as the number of bipartite graphs on n vertices is at least $2^{n^2/4}$. Thus, the convex obstacle number of an n -vertex graph with constant chromatic number can be linear in n , which asymptotically matches the linear upper bound proved by Balko, Cibulka, and Valtr [4].

3 On the Number of Graphs with Bounded Obstacle Number

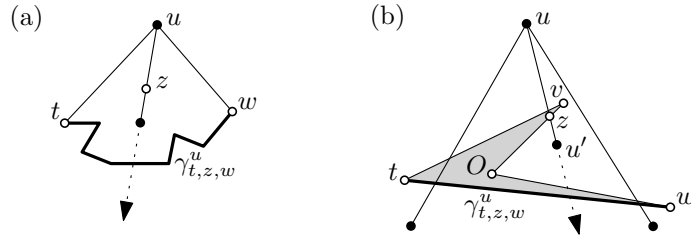
In this section we outline the proofs for Theorems 4 and 5, which provide improved upper bounds on the number of n -vertex graphs with (convex) obstacle number at most h .

3.1 Sketch of the Proof of Theorem 4

We prove that the number $f(h, n)$ of n -vertex graphs with obstacle number at most h is bounded from above by $2^{O(hn \log n)}$. We follow the approach of Mukkamala, Pach, and Pálvölgyi [18] and encode the obstacle representations using point set order types for the vertices of the graph and the obstacles. They used a bound of $O(n \log n)$ for the number of vertices of a single obstacle. We show that this bound can be reduced to $O(n)$. As a result, the upper bound on $f(h, n)$ improves by a factor of $\Theta(\log n)$ in the exponent.

An obstacle representation (D, \mathcal{O}) of a graph G is *minimal* if it uses the minimum number $|\mathcal{O}|$ of obstacles and there is no obstacle representation (D', \mathcal{O}') of G with $|\mathcal{O}'| = |\mathcal{O}|$ such that the total number of vertices in \mathcal{O}' is strictly smaller than in \mathcal{O} . Consider a graph G that admits an obstacle representation using at most h obstacles and an obstacle $O \in \mathcal{O}$. If O is a triangle, it contributes three vertices to our linear bound, which is fine. So suppose that O has four or more vertices. We want to charge the “excess” vertices to some vertex of G such that every vertex of G pays for at most a constant number of obstacle vertices.

A vertex v of a simple polygon P is *convex* if the internal angle of P at v is less than π ; otherwise v is a *reflex* vertex of P . Let v be a convex vertex of an obstacle O from \mathcal{O} and let e_1 and e_2 be the two edges of O adjacent to v . If the interior of the triangle that is spanned by e_1 and e_2 does not contain any vertex of D or \mathcal{O} , we call the vertex v *blocking*.



■ **Figure 1** (a) An illustration of the polygonal chain $\gamma_{t,z,w}^u$ used in the definition of responsibility. (b) An example where u is responsible for t and w but not for v . The vertex u' is responsible for v .

If v is blocking, then there is a non-edge e of G that crosses e_1 and e_2 , since otherwise we could reduce the number of obstacle vertices by removing the triangle spanned by e_1 and e_2 from O . If e has no other intersections with the obstacles, we say that e *forces* v . Then each convex vertex of O is forced by at least one non-edge of G . Let uu' be an edge that forces v and let z be a crossing of uu' with an edge of O incident to v . We say that u is *responsible* for v unless there are non-edges of G incident to u and forcing two other vertices t and w of O that span a polygonal chain $\gamma_{t,z,w}^u$ formed by the part of the boundary of O between t and w such that $\gamma_{t,z,w}^u$ intersects the ray \vec{uz} at some point different from z ; see Fig. 1.

With this definition of responsibility, we can show that every blocking vertex v of O has at least one vertex of G that is responsible for v and that each vertex of G is responsible for at most two blocking vertices of O . It then follows from a simple double-counting argument that O has at most $2n$ blocking vertices. A similar argument based on a different notion of responsibility is used to show that O contains at most $2r_O$ non-blocking convex vertices where r_O is the number of reflex vertices of O . Accounting for the reflex vertices is more challenging, but we can show that each reflex vertex of an obstacle can be charged to a vertex of G , so that there are no more than $2n + 5$ reflex vertices overall. Let s denote the total number of obstacle vertices, and let $r = \sum_{O \in \mathcal{O}} r_O$ denote the total number of reflex vertices. Then $s \leq r + \sum_{O \in \mathcal{O}} (2r_O + 2n) \leq 3r + |\mathcal{O}| \cdot 2n \leq (2h + 6)n + 15$.

First, it is not difficult to show that we can restrict ourselves without loss of generality to connected graphs. To bound the number $g(h, n)$ of connected n -vertex graphs with obstacle number at most h , we use a combinatorial description of point sets using order types. The *order type* of a set P of points in the plane in general position is a mapping χ that assigns $\chi(a, b, c) \in \{+1, -1\}$ to each ordered triple (a, b, c) of points from P , indicating whether a, b, c make a left turn (+1) or a right turn (-1). The order type captures which pairs of line segments spanned by P cross. Hence, for the purposes of plane straight-line embeddings of graphs, two point sets with the same order type are equivalent. The following classical result by Goodman and Pollack [14, 15] gives an estimate on the number of order types.

▶ **Theorem 11** ([14, 15]). *The number of order types on n labeled points in \mathbb{R}^2 is at most $\left(\frac{n}{2}\right)^{4(1+o(1))n} \in 2^{O(n \log n)}$.*

We encode G by the order type of all $n + s$ vertices. By Theorem 11, the number of such order types is at most $2^{\alpha(n+s) \log(n+s)}$ for some $\alpha > 0$. Since $s \leq 2hn + 6n + 15$ and $h \leq \binom{n}{2}$, we obtain $g(h, n) \in 2^{\alpha(n+2hn+6n+15) \log(n+2hn+6n+15)} \subseteq 2^{O(hn \log n)}$.

3.2 Sketch of the Proof of Theorem 5

We show that the number of graphs on n vertices that have convex obstacle number at most h is bounded from above by $2^{O(n(h+\log n))}$. That is, we prove $f_c(h, n) \in 2^{O(n(h+\log n))}$.

To this end, we give an efficient encoding for an obstacle representation (D, \mathcal{O}) of an n -vertex graph $G = (V, E)$ that uses at most h convex obstacles. The first part of the encoding is formed by the order type of V . Then, it remains to encode the obstacles and their interaction with the line segments between points from V . To do that, we use so-called radial systems. The *clockwise radial system* R of V assigns to each vertex $v \in V$ the clockwise cyclic ordering $R(v)$ of the $n - 1$ rays in D that start from v and pass through a vertex from $V \setminus \{v\}$. The order type of V uniquely determines the radial system R of V . Essentially, this also holds in the other direction: There are at most $n - 1$ order types that are compatible with a given radial system [1].

For a vertex $v \in V$ and an obstacle $O \in \mathcal{O}$, let $I_O(v)$ be the subsequence of rays in $R(v)$ that intersect O . We say that a subset I of $R(v)$ is an *interval* if there are no four consecutive elements a, b, c, d in the radial order $R(v)$ such that $a, c \in I$ and $b, d \notin I$. Since O is connected, each set $I_O(v)$ forms an interval in $R(v)$, the *blocking interval* of the pair (v, O) . By the following lemma the blocking intervals suffice to determine which edges are blocked.

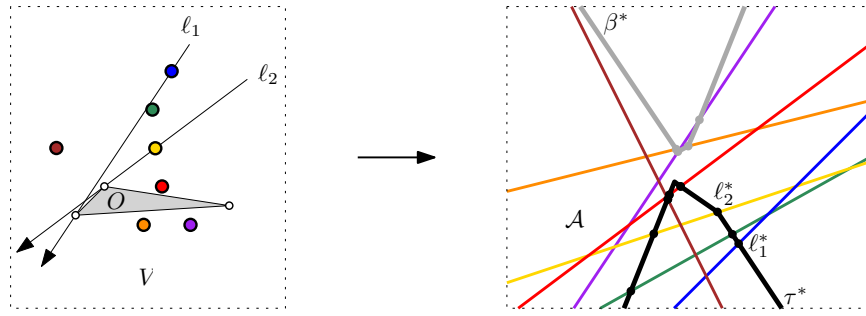
► **Lemma 12.** *For two vertices $u, v \in V$, the pair $\{u, v\}$ is a non-edge of G if and only if there is an obstacle $O \in \mathcal{O}$ such that $u \in I_O(v)$ and $v \in I_O(u)$.*

Thus it suffices to encode all blocking intervals. In the following we describe how to obtain an encoding of size $2^{O(hn)}$, which together with the order type of V yields the claimed bound on $f_c(h, n)$. In order to describe our approach it is more convenient to move to the dual setting, using the standard *projective duality transformation* that maps a point $p = (p_x, p_y) \in \mathbb{R}^2$ to the line $p^* = \{(x, y) \in \mathbb{R}^2 : y = p_x x - p_y\}$, and that maps a non-vertical line $\ell = \{(x, y) \in \mathbb{R}^2 : y = mx + b\}$ to the point $\ell^* = (m, -b)$. This map is an involution, that is, $(p^*)^* = p$ and $(\ell^*)^* = \ell$. Moreover, it preserves incidences, that is, $p \in \ell \iff \ell^* \in p^*$, and is order-preserving as a point p lies above $\ell \iff \ell^*$ lies above p^* .

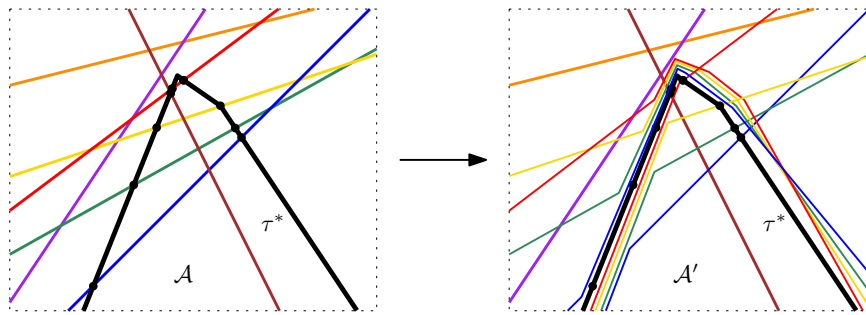
Consider the arrangement \mathcal{A} of the n lines dual to the points in V . The combinatorial structure of \mathcal{A} , that is, the sequences of intersections with other lines along each line, can be obtained from the radial system of V ; see [20]. (To identify the vertical direction and the x -order of the vertices, we add a special point very high above all other points.) Let $O \in \mathcal{O}$ be an obstacle. Define a map τ that assigns to each $x \in \mathbb{R}$ the *upper tangent* of slope x to O , that is, the line $\tau(x)$ of slope x that is tangent to O and such that O lies below $\tau(x)$; see Fig. 2. Now, consider the dual τ^* of τ , defined by $\tau^*(x) = (\tau(x))^*$. Note that, by definition, every line $\tau(x)$ passes through a vertex of the upper envelope O^+ of the convex hull of O . Consequently, each point $\tau^*(x)$ lies on a line that is dual to a vertex of O^+ . In other words, τ^* is a piece-wise linear function that is composed of line segments along the lines dual to the vertices of O^+ . The order of these line segments from left to right corresponds to primal tangents of increasing slope and, therefore, to the order of the corresponding vertices of O^+ from right to left. As primal x -coordinates correspond to dual slopes, the slopes of the line segments along τ^* monotonically decrease from left to right, and so τ^* is concave.

The primal line of a point in $v^* \cap \tau^*$, for some $v \in V$, passes through v and is an upper tangent to O . So such an intersection corresponds to an endpoint of the blocking interval $I_O(v)$. In order to obtain all endpoints of the blocking intervals, we also consider the *lower tangents* to O in an analogous manner. The corresponding function β^* is convex and consists of segments along lines dual to the vertices of the lower convex hull of O , from left to right.

It remains to compactly encode the intersections of τ^* with \mathcal{A} . To do so, we apply a result by Knuth [17]; see also the description by Felsner and Valtr [9]. A set \mathcal{A}' of biinfinite curves in \mathbb{R}^2 forms an *arrangement of pseudolines* if each pair of curves from \mathcal{A}' intersects in a unique point, which corresponds to a proper, transversal crossing. A curve α is a *pseudoline*



■ **Figure 2** An example of a vertex set V with its dual line arrangement \mathcal{A} . Points and lines of the same color are dual to each other. The curve τ^* for the upper envelope O^+ of the obstacle O is denoted black and the curve β^* for the lower envelope O^- of the obstacle O is denoted gray.



■ **Figure 3** Splitting the n lines of \mathcal{A} into two parts so that τ^* is a pseudoline with respect to the resulting arrangement \mathcal{A}' of at most $2n$ pseudolines.

with respect to \mathcal{A}' if α intersects each curve in \mathcal{A}' at most once. Let γ be a curve that intersects each curve from \mathcal{A}' in a finite number of points. The *cutpath* of γ in \mathcal{A}' is the sequence of intersections of γ with the pseudolines from \mathcal{A}' along γ .

► **Theorem 13** ([17]). *If α is a pseudoline with respect to an arrangement \mathcal{A}' of n pseudolines, then there are at most $O(3^n)$ cutpaths of α in \mathcal{A}' .*

Using Theorem 13, we can now estimate the number of cutpaths of τ^* with respect to \mathcal{A} .

► **Lemma 14.** *There are at most $O(9^n)$ cutpaths of τ^* in \mathcal{A} .*

Proof. If τ^* was a pseudoline with respect to \mathcal{A} , then we were done by Theorem 13. However, τ^* is not a pseudoline with respect to \mathcal{A} in general because a vertex $v \in V$ can have two tangents to O^+ , in which case τ^* intersects v^* twice. Therefore, we split the lines of \mathcal{A} that correspond to vertices vertically above O into two parts, by removing the part of each line in a close neighborhood of its leftmost intersection with τ^* . Then both parts have only one intersection with τ^* . Each of the two stubs can be extended by following τ^* and crossing whatever needs to be crossed so that both together with all other split curves, the remaining lines from \mathcal{A} , and τ^* form a pseudoline arrangement \mathcal{A}' of at most $2n + 1$ pseudolines; see Fig. 3. Then, by Theorem 13, there are at most $O(3^{2n}) = O(9^n)$ cutpaths of τ^* in \mathcal{A}' . ◀

To summarize, we encode the obstacle representation (D, \mathcal{O}) of G by first encoding the order type of V . By Theorem 11, there are at most $2^{O(n \log n)}$ choices for the order type of a set of n points. The order type of V determines the arrangement \mathcal{A} of n lines that are dual to the points from V . By Lemma 12, it suffices to encode the endpoints of the blocking

intervals $I_O(v)$ for every $v \in V$ and $O \in \mathcal{O}$, which are defined using the radial system of V that is also determined by the order type of V . For each obstacle $O \in \mathcal{O}$, the endpoints of all the intervals $I_V(O)$ are determined by the cutpath of the curve τ^* in \mathcal{A} constructed for the upper tangents of O and the analogous curve β^* constructed for the lower tangents of O . By Lemma 14, there are at most $O(9^n)$ cutpaths of τ^* in \mathcal{A} and the same estimate holds for β^* . This gives at most $O((9^n)^2) = O(81^n)$ possible ways how to encode a single obstacle. Altogether, we thus obtain $f_c(h, n) \in 2^{O(n \log n)} \cdot O(81^{hn}) \subseteq 2^{O(n \log n + hn)}$.

4 A Lower Bound on the Obstacle Number of Drawings

Here, we prove Theorem 7 by showing that there is a constant $\delta > 0$ such that, for every n , there exists a graph G on n vertices and a drawing D of G such that $\text{obs}(D) \geq \delta \cdot n^2$. The proof is constructive. We first assume that n is even.

A set of points in the plane is a *cup* if all its points lie on the graph of a convex function. Similarly, a set of points is a *cap* if all its points lie on the graph of a concave function. For an integer $k \geq 3$, a convex polygon P is *k-cap free* if no k vertices of P form a cap of size k . Note that P is *k-cap free* if and only if it is bounded from above by at most $k - 1$ segments (edges of P). We use $e(P)$ to denote the leftmost edge bounding P from above; see Fig. 4(a).

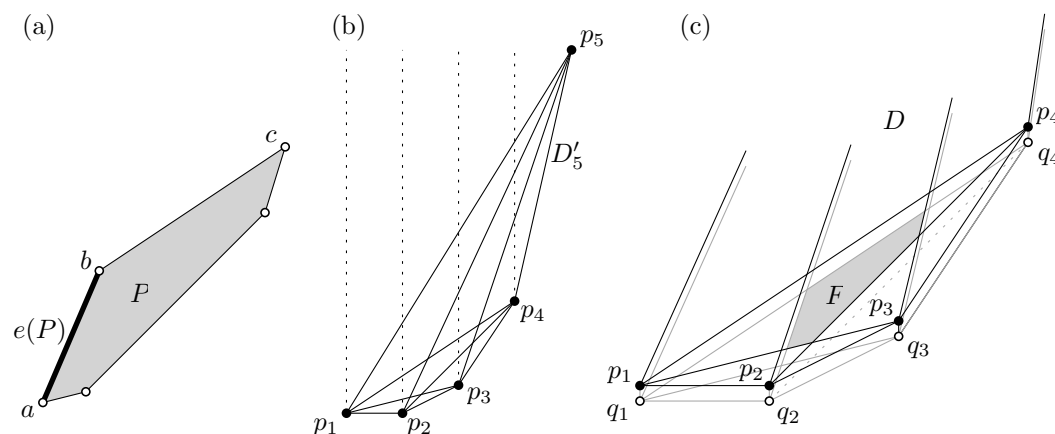


Figure 4 (a) A 4-cap free polygon P that is not 3-cap free. (b) An example of the drawing D'_m for $m = 5$. If the point p_m is chosen sufficiently high above C_{m-1} , then each line segment $\overline{p_i p_m}$ with $i < m$ is very close to the vertical line containing p_i and thus all faces of D'_m will be 4-cap free. (c) An example of a part of the drawing D with an example of a face F of D . The line segments that are not in D'_m are drawn grey and the non-edge from X is dotted. For better readability of the picture, edges $p_i q_j, i \neq j$, are not drawn.

Let $m = n/2$. First, we inductively construct a set C_m of m points p_1, \dots, p_m in the plane that form a cup and have x -coordinates $x(p_i) = i$. We let $C_1 = \{(1, 0)\}$ and $C_2 = \{(1, 0), (2, 0)\}$. Now, assume that we have already constructed a set $C_{m-1} = \{p_1, \dots, p_{m-1}\}$ for some $m \geq 3$. Let D'_{m-1} be the drawing of the complete graph with vertices p_1, \dots, p_{m-1} . We choose a sufficiently large number y_m , and we let p_m be the point (m, y_m) and $C_m = C_{m-1} \cup \{p_m\}$. We also let D'_m be the drawing of the complete graph with vertices p_1, \dots, p_m . The number y_m is chosen large enough so that the following three conditions are satisfied:

1. for every $i = 1, \dots, m - 1$, every intersection point of two line segments spanned by points of C_{m-1} lies on the left side of the line $p_i p_m$ if and only if it lies to the left of the vertical line $x = i$ containing the point p_i ,

11:10 Bounding and Computing Obstacle Numbers of Graphs

2. if F is a 4-cap free face of D'_m that is not 3-cap free, then there is no point p_i below the (relative) interior of $e(F)$,
3. no crossing of two edges of D'_m lies on the vertical line containing some point p_i .

Note that choosing the point p_m is possible as choosing a sufficiently large y -coordinate y_m of p_m ensures that for each i , all the intersections of the line segments $p_i p_m$ with line segments of D'_{m-1} lie very close to the vertical line $x = i$ containing the point p_i . Furthermore, the construction implies that no line segment of D'_m is vertical and that no point is an interior point of more than two line segments of D'_m . The drawing D'_m also satisfies the next claim.

▷ **Claim 15.** Each inner face of D'_m is a 4-cap free convex polygon.

For some (small) $\varepsilon > 0$ and every $i \in \{1, \dots, m\}$, where $m = n/2$, we let q_i be the point $(i, y_i - \varepsilon)$. That is, q_i is a point slightly below p_i . We choose ε sufficiently small so that decreasing ε to any smaller positive real number does not change the combinatorial structure of the intersections of the line segments spanned by the n points $p_1, \dots, p_m, q_1, \dots, q_m$. We let $D = D_n$ be the drawing with n vertices $p_1, \dots, p_m, q_1, \dots, q_m$ containing all the line segments between two vertices, except the line segments $\overline{q_i q_j}$ where i and j are both even.

We have to show that at least quadratically many obstacles are needed to block all non-edges of G in D . Let X be the set of line segments $\overline{q_i q_j}$ where both i and j are even. Note that each line segment from X corresponds to a non-edge of G and that $|X| = \binom{\lfloor n/4 \rfloor}{2} \geq n^2/40$ for a sufficiently large n . The proof of the following claim is based on Claim 15.

▷ **Claim 16.** Every face of D is intersected by at most two line segments from X .

Now, let \mathcal{O} be a set of obstacles such that D and \mathcal{O} form an obstacle representation of G . Then each non-edge of G corresponds to a line segment from X that is intersected by some obstacle from \mathcal{O} . Each obstacle O from \mathcal{O} lies in some face F_O of D as it cannot intersect an edge of G . Thus, O can intersect only line segments from X that intersect F_O . It follows from Claim 16 that each obstacle from \mathcal{O} intersects at most two line segments from X . Since $|X| \geq n^2/40$, we obtain $|\mathcal{O}| \geq |X|/2 \geq n^2/80$. Consequently, $\text{obs}(D) \geq n^2/80$, which finishes the proof for n even. If n is odd, we use the above construction for $n - 1$ and add an isolated vertex to it.

5 Obstacle Number is FPT Parameterized by Vertex Cover Number

As our first step towards a fixed-parameter algorithm, we need to establish a bound on the obstacle number of every graph with bounded vertex cover number.

► **Lemma 17.** *Let G be a graph with vertex cover number k . Then G admits an obstacle representation with $1 + \binom{k}{2} + k \cdot 2^k$ obstacles.*

Proof Sketch. We construct an obstacle representation for G as follows. We place the vertices in the vertex cover X somewhere in a central area, and dedicate at most $\binom{k}{2}$ obstacles to handle the visibilities between these. Next, we partition the vertices outside of X into at most 2^k equivalence classes called *types* based on the following equivalence: two vertices $a, b \notin X$ have the same type if and only if they have the same neighborhood. We place these vertices on a line ℓ sufficiently far from the other vertices, and group them by type. We then use a single obstacle to block the pairwise visibilities of vertices on ℓ , and at most $k \cdot 2^k$ obstacles to block visibilities between each type and each vertex in X where necessary. ◀

Our proof will also rely on a Ramsey-type argument based on the following formulation of Ramsey's theorem [19]:

► **Theorem 18** ([19]). *There exists a function $\text{Ram}: \mathbb{N}^2 \rightarrow \mathbb{N}$ with the following property. For each pair of integers q, s and each clique H of size at least $\text{Ram}(q, s)$ such that each edge has a single label (color) out of a set of q possible labels, it holds that H must contain a subclique K of size at least s such that every edge in K has the same label.*

We can now proceed towards a proof of Theorem 8, which establishes the fixed-parameter tractability of computing the obstacle number with respect to the vertex cover number.

Proof Sketch for Theorem 8. Let X be a computed vertex cover of size at most k in the input graph G . If $h > 1 + \binom{k}{2} + k \cdot 2^k$ we immediately output “Yes” by Lemma 17, and so for the rest of the proof, we proceed assuming that $h \leq 1 + \binom{k}{2} + k \cdot 2^k$. We also note that if G contains only types of size bounded by a fixed, sufficiently large function of k (which we will hereinafter denote $\text{typesize}(k)$), then the instance can be solved by brute force. Hence, we assume that G contains at least one type T with more than $\text{typesize}(k)$ vertices.

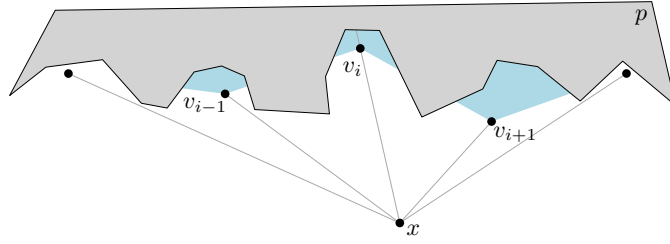
Our proof strategy will be to show that under these conditions, G is a **Yes**-instance (i.e., admits a representation with at most h obstacles) if and only if the graph G'' obtained by deleting an arbitrary vertex from T is a **Yes**-instance; in other words, that we may prune a vertex from T . This claim would immediately yield fixed-parameter tractability of the problem: one could iterate the procedure of computing a vertex cover for the input graph, checking whether the types are sufficiently large, and based on that check one either brute-forces the problem or restarts on a graph that contains one fewer vertex (notice that the number of restarts is upper-bounded by n).

As a first step, let us consider a hypothetical “optimal” solution S for (G, h) , i.e., S is an obstacle representation of G with the minimum number of obstacles. It is easy to observe that simply using the same placement of obstacles for the graph G'' obtained by *removing* a single vertex from T yields a desired solution S' for (G'', h) . It is the converse direction that is the difficult one: for that, it suffices to establish the equivalent claim that the graph G' obtained by *adding* a vertex to T admits an obstacle representation S' with the same number of obstacles as S .

Given S , we will now consider an auxiliary edge-labeled clique H whose vertices are precisely the vertices in T , which can be assumed to be ordered by \prec in an arbitrary but fixed way. For each pair of vertices $a, b \in T$ where $a \prec b$, the a - b edge in H is labeled by the first obstacle encountered when traversing the line segment \overline{ab} . Crucially, by Theorem 18 and the selection of $\text{typesize}(k) \geq \text{Ram}(h, 10k^3 \cdot 2^k)$, H must contain a subclique K of size at least $\text{cliquesize}(k) = 10k^3 \cdot 2^k$ such that each edge in K has the same label, say p .

Intuitively, our aim in the rest of the proof will be to show that the obstacle p can be “safely” extended towards some vertex z in K , where by safely we mean that it neither intersects another obstacle nor blocks the visibility of an edge; this extension will either happen directly from S , or from a slightly altered version of S . Once we create such an extension, it will be rather straightforward to show that p can be shaped into a tiny “comb-like” slot for a new vertex z' next to z which will have the same visibilities (and hence neighborhood) as z , which means we have constructed a solution S' for G' as desired.

To this end, we show that almost all vertices in K must be placed in a system of *crevices* formed by p (these can be imagined as “openings” in p , each containing a specific *crevice vertex* from K ; see Fig. 5). By a sequence of claims, we can then restrict our attention to a still sufficiently large subsystem of these crevices which are *good*, meaning that they contain no obstacle other than p , no vertex from the vertex cover, and where visibilities between pairs of vertices outside the crevice would not be obstructed by extending p towards the crevice vertex. However, other vertices may still be present in good crevices, and so such an extension could still obstruct the visibility of vertices inside the crevice.



■ **Figure 5** The crevices of obstacle p are the light blue regions containing the vertices $\dots, v_{i-1}, v_i, v_{i+1}, \dots$ of K .

To deal with this final issue and extend p to a crevice vertex as required to conclude the proof, we will seek out a crevice which is not just good but also *perfect*, meaning that it contains no vertex other than the crevice vertex. While S itself need not contain a perfect crevice, in the last part of the proof we use a swapping argument on S to argue the existence of an equivalent solution which is guaranteed to contain one. ◀

6 NP-Hardness of Deciding Whether a Given Obstacle is Enough

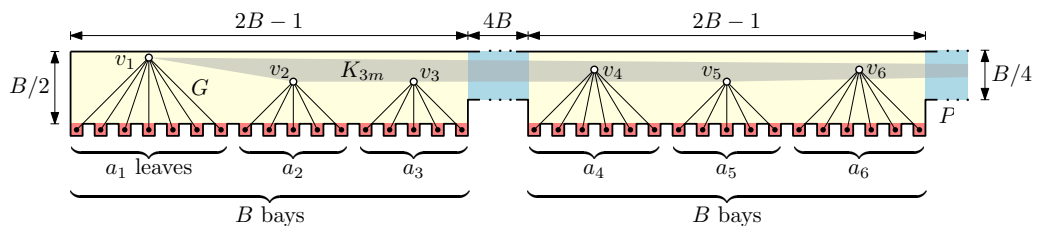
In this section, we investigate the complexity of the obstacle representation problem for the case that a single (outside) obstacle is given.

► **Theorem 9.** *Given a graph G and a simple polygon P , it is NP-hard to decide whether G admits an obstacle representation using P as (outside-) obstacle.*

Proof Sketch. We reduce from 3-PARTITION, which is NP-hard [11]. An instance of 3-PARTITION is a multiset S of $3m$ positive integers a_1, \dots, a_{3m} , and the question is whether S can be partitioned into m groups of cardinality 3 such that the numbers in each group sum up to the same value B , which is $(\sum_{i=1}^{3m} a_i)/m$. The problem remains NP-hard if B is polynomial in m and if, for each $i \in [3m]$, it holds that $a_i \in (B/4, B/2)$.

Given a multiset S , we construct a graph G and a simple polygon P with the property that the vertices of G can be placed in P such that their visibility graph with respect to P is isomorphic to G if and only if S is a yes-instance of 3-PARTITION.

Let G be a clique with vertices v_1, \dots, v_{3m} where, for $i \in [3m]$, *clique vertex* v_i is connected to a_i leaves $\ell_{i,1}, \dots, \ell_{i,a_i}$. The polygon P (see Fig. 6) is an orthogonal polygon with m groups of B bays (red in Fig. 6) that are separated from each other by corridors (of height $B/4$ and width $4B$; light blue in Fig. 6). Each bay is a unit square; any two consecutive bays are one unit apart. The height of P (including the bays) is $B/2 + 1$. Note that the sizes of G and P are polynomial in m . ◀



■ **Figure 6** Idea of the reduction from 3-PARTITION. The drawing is not to scale.

References

- 1 Oswin Aichholzer, Jean Cardinal, Vincent Kusters, Stefan Langerman, and Pavel Valtr. Reconstructing point set order types from radial orderings. *Internat. J. Comput. Geom. Appl.*, 26(3-4):167–184, 2016. doi:10.1142/S0218195916600037.
- 2 Hannah Alpert, Christina Koch, and Joshua D. Laison. Obstacle numbers of graphs. *Discrete Comput. Geom.*, 44(1):223–244, 2010. doi:10.1007/s00454-009-9233-8.
- 3 Martin Balko, Steven Chaplick, Robert Ganian, Siddharth Gupta, Michael Hoffmann, Pavel Valtr, and Alexander Wolff. Bounding and computing obstacle numbers of graphs, 2022. arXiv:2206.15414.
- 4 Martin Balko, Josef Cibulka, and Pavel Valtr. Drawing graphs using a small number of obstacles. *Discrete Comput. Geom.*, 59(1):143–164, 2018. doi:10.1007/s00454-017-9919-2.
- 5 Jean Cardinal and Udo Hoffmann. Recognition and complexity of point visibility graphs. *Discrete & Computational Geometry*, 57(1):164–178, January 2017. doi:10.1007/s00454-016-9831-1.
- 6 Steven Chaplick, Fabian Lipp, Ji-won Park, and Alexander Wolff. Obstructing visibilities with one obstacle. In Yifan Hu and Martin Nöllenburg, editors, *Proc. 24th Int. Symp. Graph Drawing & Network Vis. (GD)*, volume 9801 of *LNCS*, pages 295–308. Springer, 2016. doi:10.1007/978-3-319-50106-2_23.
- 7 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 8 Vida Dujmović and Pat Morin. On obstacle numbers. *Electr. J. Comb.*, 22(3):#P3.1, 2015. doi:10.37236/4373.
- 9 Stefan Felsner and Pavel Valtr. Coding and counting arrangements of pseudolines. *Discrete Comput. Geom.*, 46(3):405–416, 2011. doi:10.1007/s00454-011-9366-4.
- 10 Oksana Firman, Philipp Kindermann, Jonathan Klawitter, Boris Klemz, Felix Klesen, and Alexander Wolff. Outside-obstacle representations with all vertices on the outer face. In Patrizio Angelini and Reinhard von Hanxleden, editors, *Proc. 30th Int. Symp. Graph Drawing & Network Vis. (GD'22)*, 2022. arXiv:2202.13015.
- 11 Michael R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.*, 4(4):397–411, 1975. doi:10.1137/0204035.
- 12 Subir Kumar Ghosh and Bodhayan Roy. Some results on point visibility graphs. *Theoretical Computer Science*, 575:17–32, 2015. doi:10.1016/j.tcs.2014.10.042.
- 13 John Gimbel, Patrice Ossona de Mendez, and Pavel Valtr. Obstacle numbers of planar graphs. In Fabrizio Frati and Kwan-Liu Ma, editors, *International Symposium on Graph Drawing and Network Visualization*, volume 10692 of *LNCS*, pages 67–80. Springer, 2017. doi:10.1007/978-3-319-73915-1_6.
- 14 Jacob E. Goodman and Richard Pollack. Upper bounds for configurations and polytopes in \mathbb{R}^d . *Discrete Comput. Geom.*, 1(3):219–227, 1986. doi:10.1007/BF02187696.
- 15 Jacob E. Goodman and Richard Pollack. Allowable sequences and order types in discrete and computational geometry. In *New Trends in Discrete and Computational Geometry*, volume 10 of *Algorithms and Combinatorics*, pages 103–134. Springer, Berlin, 1993. doi:10.1007/978-3-642-58043-7_6.
- 16 Matthew P. Johnson and Deniz Sariöz. Representing a planar straight-line graph using few obstacles. In *Proc. 26th Canadian Conf. Comput. Geom. (CCCG)*, pages 95–99, 2014. URL: <http://www.cccg.ca/proceedings/2014/papers/paper14.pdf>.
- 17 Donald E. Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes Comput. Sci.* Springer, Heidelberg, Germany, 1992. doi:10.1007/3-540-55611-7.
- 18 Padmini Mukkamala, János Pach, and Dömötör Pálvölgyi. Lower bounds on the obstacle number of graphs. *Electr. J. Comb.*, 19(2):#P32, 2012. doi:10.37236/2380.
- 19 Frank P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc. (2)*, 30(4):264–286, 1929. doi:10.1112/plms/s2-30.1.264.
- 20 Emo Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Inform. Process. Lett.*, 20:167–171, 1985. doi:10.1016/0020-0190(85)90044-4.

Computing NP-Hard Repetitiveness Measures via MAX-SAT

Hideo Bannai ✉ 

Tokyo Medical and Dental University, Japan

Keisuke Goto ✉ 

Independent Researcher, Tokyo, Japan

Masakazu Ishihata ✉

NTT Communication Science Laboratories, Kyoto, Japan

Shunsuke Kanda ✉ 

Independent Researcher, Tokyo, Japan

Dominik Köppl ✉ 

Tokyo Medical and Dental University, Japan

Takaaki Nishimoto ✉

RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Abstract

Repetitiveness measures reveal profound characteristics of datasets, and give rise to compressed data structures and algorithms working in compressed space. Alas, the computation of some of these measures is NP-hard, and straight-forward computation is infeasible for datasets of even small sizes. Three such measures are the smallest size of a string attractor, the smallest size of a bidirectional macro scheme, and the smallest size of a straight-line program. While a vast variety of implementations for heuristically computing approximations exist, exact computation of these measures has received little to no attention. In this paper, we present MAX-SAT formulations that provide the first non-trivial implementations for exact computation of smallest string attractors, smallest bidirectional macro schemes, and smallest straight-line programs. Computational experiments show that our implementations work for texts of length up to a few hundred for straight-line programs and bidirectional macro schemes, and texts even over a million for string attractors.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases repetitiveness measures, string attractor, bidirectional macro scheme

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.12

Supplementary Material *Software (Source Code)*: <https://github.com/kg86/satcomp>
archived at `swh:1:dir:9f4f16f948f46118492771d403a0ca880a7742ab`

Funding *Hideo Bannai*: Supported by JSPS KAKENHI Grant Number JP20H04141.

Dominik Köppl: Supported by JSPS KAKENHI Grant Numbers JP21H05847 and JP21K17701.

1 Introduction

Text compression is a fundamental topic in computer science with countless practical applications. *Dictionary compression* is a type of text compression where the original input is transformed into a sequence of elements taken from a dictionary, where the dictionary is usually constructed in some way from the input. Due to the advent of *highly repetitive* datasets such as multiple genome sequences from the same species or versioned document collections (e.g., Wikipedia, GitHub), dictionary compression methods have recently (re)gained massive attention since they can better capture more widespread repetitions in such data compared to statistical compression methods [30], and further allow space-efficient full-text



© Hideo Bannai, Keisuke Goto, Masakazu Ishihata, Shunsuke Kanda, Dominik Köppl, and Takaaki Nishimoto;

licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 12; pp. 12:1–12:16



Leibniz International Proceedings in Informatics

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

indices to be built [31]. Some well known methods that fall in this category are Lempel–Ziv 76/77 factorization based methods [20, 23, 43], grammar-based compression such as LZ78 [44], Re-Pair [22], SEQUITUR [34], LCA [38], LZD [11], and methods involving bidirectional referencing, such as the run-length encoded Burrows–Wheeler transform (RLBWT) [25], and more recently, lpccomp [10], plpccomp [9], lexcomp [32], a method by Russo et al. [36], and LZRR [35].

A vital issue in evaluating and comparing these various methods is to understand how well they can compress a given input compared to the “optimum”. While the theoretically smallest representation (aka Kolmogorov complexity) is incomputable [24], Kempa and Prezza [16] regarded the output sizes of these methods as *repetitiveness measures* and characterized them with respect to the new notion of *string attractors*. Namely, they showed that for any input text, the size of the smallest string attractor is a lower bound for the output sizes of all known dictionary compressors. Since then, relations between these various repetitiveness measures have been heavily investigated [2, 4, 14, 17, 19, 30, 32].

In this paper, we consider three such repetitiveness measures: the size γ of the smallest string attractor, the size g of the smallest straight-line program (SLP) [13], and the size b of the smallest bidirectional macro scheme (BMS) [42], all of which are known to be NP-hard to compute [16, 39, 42]. Thus, any efficient dictionary compression algorithm can (most likely) merely compute approximations of γ , b , or g . Although for any text, the relation $\delta \leq \gamma \leq b \leq z \leq g$ is known, where δ [19] and z [23] are repetitiveness measures known to be computable in linear time (cf. [7, Lemma 5.7] for δ and [8] for z), the gap between the measures can be quite large; string families giving a logarithmic factor gap are known for each pair of measures [2, 30]. Since the sizes of some recent data structures such as [7, 33], depend on these repetitiveness measures, their exact sizes are crucial knowledge.

While there exist a vast variety of approximation algorithms for computing smallest BMSs and grammars as mentioned above, development of exact algorithms have received very little to almost no attention. For string attractors, the results of Kempa et al. [15] imply a straightforward $O(n2^n)$ time algorithm. For the smallest grammar, Casel et al. [6, Theorem 13] show an $O^*(3^n)^1$ time algorithm. However, we are unaware of any non-trivial implementations or empirical evaluations for computing these measures. In fact, the only publicly available implementation we could find was a straight-forward Python script to compute γ by Michael S. Branicky [12].

The main contribution of this paper is to present MAX-SAT formulations [3] for computing the smallest string attractor, BMS, and SLP, thereby providing the first non-trivial implementation for exact computation of the measures γ , b , and g . The rationale for this approach is that although MAX-SAT is NP-hard, there are highly optimized solvers whose performance has made incredible progress in recent years. These solvers can cope with very large instances and can be leveraged, provided that suitable encodings can be designed [18]. While straight-forward (non-MAX-SAT) implementations become infeasible even for very small text lengths (e.g. 40), computational experiments show that our implementations work for texts of length up to a few hundred for b , g , and even more than 1 million for γ . Since our addressed problems are all NP-hard, there is perhaps little hope for our implementations to obtain exact solutions for larger but practically interesting datasets. Nevertheless, we believe they can make significant impact as a tool for analyzing these repetitiveness measures. We stress that our solutions not only report the sizes γ , b and g , but also give valid instances having exactly these sizes (e.g., an SLP that has size g). It may therefore be possible to

¹ The abstract of [6] mentions $O(3^n)$ while the statement of the theorem is $O^*(3^n)$.

improve compression heuristics by studying some of these optimal instances on smaller input strings. As an example application, we analyzed the recently introduced notion of *sensitivity* [1] of γ by conducting an exhaustive computation of γ for strings up to certain lengths. From these computations, we were able to discover a family of strings that exhibit a multiplicative sensitivity of 2.5, improving the previously known lower bound of 2.0 [1].

Related Work

The exact values for γ , b , and g have been characterized only for a few families of strings. For standard Sturmian words, $\gamma = 2$ [26] and $b = O(1)$ since the RLBWT has constant size [27] and can be regarded as a BMS. For the n th Thue–Morse word, $\gamma = 4$ for $n \geq 4$ [21], and $b = n + 2$ for $n \geq 2$ [2]. For the n th Fibonacci word, $g = n$ [28]. The smallest attractor sizes of automatic sequences have also been studied [40].

2 Preliminaries

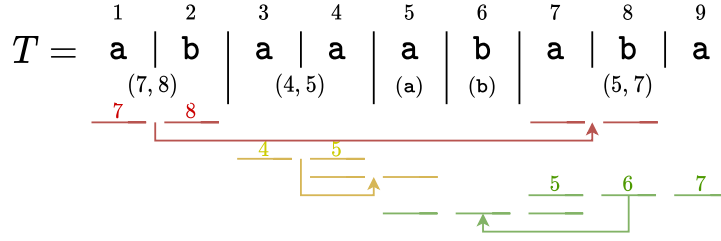
Let Σ be a set of σ symbols called the *alphabet*, and let Σ^* denote the set of strings over Σ . Given a string T , if $T = xyz$ for strings x, y, z , then x, y, z are respectively called a *prefix*, *substring*, and *suffix* of T . They are called *proper* if they are not equal to T . The length of T is denoted by $|T|$. For any $i \in [1, |T|]$, let $T[i]$ denote the i th symbol of T , i.e., $T = T[1] \cdots T[|T|]$. For any $1 \leq i \leq j \leq |T|$, let $T[i..j] = T[i] \cdots T[j]$ and $T[i..j) = T[i] \cdots T[j - 1]$.

For the rest of this paper, we fix a string T , and let $n := |T|$ denote its length. Further, we assume that each symbol of Σ appears in T . Let $occ(P) = \{i \mid T[i..i + |P| - 1] = P, 1 \leq i \leq n - |P| + 1\}$ be the set of starting positions of all occurrences of a substring P in T , and let $cover(P) = \{i + k - 1 \mid i \in occ(P), 1 \leq k \leq |P|\}$ be the set of all text positions covered by all occurrences of P in T .

A set of positions $\Gamma \subseteq [1, n]$ is a *string attractor* [16] of T if every substring P of T has an occurrence in T that contains an element of Γ , that is, $\Gamma \cap cover(P) \neq \emptyset$. We denote the size of the smallest string attractor of T by γ . For example, $[1, n]$ is a trivial string attractor. $\{1, 2, 3\}$ is a (smallest) string attractor of $T = \text{banana}$. (See also Figure 2)

A *straight-line program* (SLP) [13] is a grammar in Chomsky normal form whose language consists solely of T . In other words, (1) each production rule is of the form $X \rightarrow X_\ell X_r$ or $X \rightarrow c$, where X_ℓ, X_r are non-terminals and $c \in \Sigma$, (2) there is exactly one such production rule for any given non-terminal symbol X , and (3) there is a start symbol whose iterative expansion finally leads to T . The *size* of an SLP is the number of its production rules, or equivalently (assuming that each non-terminal is used at least once), the number of distinct non-terminals. We denote the size of the smallest SLP that produces T by g . For example, the set of production rules $\{X_9 \rightarrow X_6 X_8, X_8 \rightarrow X_7 X_7, X_7 \rightarrow X_1 X_3, X_6 \rightarrow X_4 X_5, X_5 \rightarrow X_3 X_3, X_4 \rightarrow X_3 X_1, X_3 \rightarrow X_1 X_2, X_2 \rightarrow \mathbf{b}, X_1 \rightarrow \mathbf{a}\}$ is an SLP of size 9 for $T = \text{abaababaabaab}$. See also Figure 3.

A *bidirectional macro scheme* (BMS) [42] of size m representing T , is a factorization $T = F_1 \cdots F_m$, where each factor (or phrase) is a single symbol (which we call a ground phrase), or, is encoded as a pair of integers (i, j) indicating that it references (i.e., is a copy of) substring $T[i..j]$. A BMS is said to be *valid*, if T can be reconstructed from the representation of such a factorization, i.e., the implied references of each symbol in a non-ground phrase is acyclic, and eventually leads to a ground phrase. We denote the size of the smallest valid BMS that represents T by b . Figure 1 shows a valid BMS $(7, 8), (4, 5), \mathbf{a}, \mathbf{b}, (5, 7)$ representing the string abaaababa . For example, the \mathbf{a} at position 9 references position 7, which in turn references position 5, a ground phrase.



■ **Figure 1** A bidirectional macro scheme (BMS) of $T = \text{abaaababa}$. The figure depicts the BMS $(7, 8), (4, 5), a, b, (5, 7)$. The reference of each non-ground phrase is visualized by an arrow. The phrase references imply a reference for each symbol in non-ground phrases.

The satisfiability (SAT) problem asks for an assignment of variables that satisfies a given Boolean formula [3, 18]. The input formula is usually given in *conjunctive normal form* (CNF), which consists of a conjunction of clauses, and each clause is a disjunction of literals. A *literal* is a Boolean variable or its negation. In this form, the given formula is satisfied if and only if all the clauses (which we will sometimes call constraints) are satisfied. The *size* of a CNF is the sum of the literals in all clauses.

A *maximum satisfiability* (MAX-SAT) problem is an extension of SAT, where two types of clauses, *hard* and *soft*, are considered [3]. A solution to a MAX-SAT instance is a truth assignment of the variables such that the number of satisfied soft clauses is maximized under the restriction that all hard clauses must be satisfied.

We will use 1 to denote true, and 0 to denote false. Furthermore, for a set $\{v_i\}_{i=1}^k$ of Boolean variables, cardinality constraints of the form $\sum_{i=1}^k v_i \leq 1$ are known as *atmost-one constraints*. Although a straightforward encoding has size $\Theta(k^2)$, $O(k)$ size encodings are known [41]. Constraints of the form $\sum_{i=1}^k v_i = 1$ can be encoded using a combination of an atmost-one constraint and a simple disjunction of all the variables (i.e., at least-one) and thus can also be encoded in $O(k)$ size.

3 Reductions to MAX-SAT

In what follows, we present our encodings for the aforementioned problems. Common to all encodings is the idea that we have a Boolean variable p_i for each text position $i \in [1, n]$, which counts, when set to true, an element of a string attractor, a non-terminal (actually, to be precise, a factor in a grammar parsing) of an SLP, or a phrase of a BMS. Since our goal is to have as few p_i 's set to true as possible, our soft clauses have the form $D_i = \neg p_i$ for $i \in [1, n]$. Consequently, all our encodings have the same number of soft clauses, and only differ in how the hard clauses are defined.

3.1 Smallest String Attractor as MAX-SAT

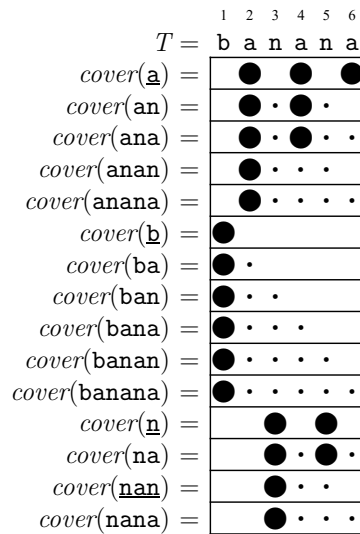
We start with a simple encoding based on the definition of string attractors. Subsequently, we utilize an observation similar to but slightly more generalized than that made in [15], in order to reduce the size of hard clauses.

3.1.1 Simple Encoding

Our idea is to design a CNF so that a MAX-SAT solution will encode a string attractor Γ , where $p_i = 1$ if and only if position i is an element of Γ (i.e., $\Gamma = \{i \mid 1 \leq i \leq n, p_i = 1\}$). Let \mathcal{S}_T denote the set of all non-empty substrings of T , i.e., $\mathcal{S}_T = \{T[i..j] \mid 1 \leq i \leq j \leq n\}$.

For each substring S of \mathcal{S}_T , we define a hard clause $C_S = \bigvee_{i \in \text{cover}(S)} p_i$. (See Figure 2 for an example.) By the definition of $\text{cover}(S)$, the set Γ corresponding to any truth assignment for p_i will be a string attractor if and only if all hard clauses C_S are satisfied. Since our soft clauses have the form $D_i = \neg p_i$ for $i \in [1, n]$, the soft clauses ensure that the MAX-SAT solution minimizes the number of p_i 's being true. Thus, we can obtain the smallest string attractor by solving the MAX-SAT on C_S and D_i .

Each hard clause C_S has size $|\text{cover}(S)| = O(n)$. Since there are $O(n^2)$ substrings, the number of hard clauses is $O(n^2)$. Hence, the total size of the CNF is $O(n^3)$. In the next subsection, we reduce the size to $O(n^2)$.



■ **Figure 2** String $T = \text{banana}$ and the positions that each distinct substring of T covers. We list all distinct substrings of T on the left hand side, and show on the right hand side their covers. A dot at position k in the row for substring S indicates that k is included in $\text{cover}(S)$ (i.e. is covered by S), and a large dot indicates $k \in \text{occ}(S)$. Underlined substrings are minimal substrings of T . For example, $\text{cover}(\underline{an}) = \{2, 3, 4, 5\}$. The clause defined for \underline{an} in our encoding is $C_{\underline{an}} = p_2 \vee p_3 \vee p_4 \vee p_5$.

3.1.2 Reducing CNF Clauses via Minimal Substrings

We can reduce the number of hard clauses in our CNF by considering only members of \mathcal{S}_T that are *minimal substrings*². A substring S of string T is called a minimal substring of T if all proper substrings of S occur more often than S in T (i.e., $|\text{occ}(S[i..j])| > |\text{occ}(S)|$ for every proper substring $S[i..j]$ of S). By the definition of minimal substrings, the following lemma holds.

► **Lemma 1.** *For every non-minimal substring S of T , there is a minimal substring S_{\min} of S with $\text{cover}(S_{\min}) \subseteq \text{cover}(S)$.*

² Kempa et al. [15] use a similar idea when reducing the problem to set cover. Their formulation can be regarded as considering only right-minimal substrings (i.e., $|\text{occ}(S[1..|S| - 1])| > |\text{occ}(S)|$), while we consider a potentially smaller subset requiring both right-minimality and left-minimality. For texts in the Calgary corpus, we observed that the difference between minimal and right-minimal substrings can result in a difference as large as 50 times in their total lengths (prog and trans), i.e., the total size of hard clauses.

Proof. Because S is not minimal, it has substrings that have the same number of occurrences as S . Let $S_{\min} = S[e..e+|S_{\min}|-1]$ be one of these substrings that is minimal, for some e . Then by definition, for each occurrence $i_{\min} \in \text{occ}(S_{\min})$ of S_{\min} , there exists an occurrence $i \in \text{occ}(S)$ of S such that $i = i_{\min} - e + 1$. $\{i_{\min}, i_{\min} + 1, \dots, i_{\min} + |S_{\min}| - 1\} \subseteq \{i, i + 1, \dots, i + |S| - 1\}$, and hence, $\text{cover}(S_{\min}) \subseteq \text{cover}(S)$. \blacktriangleleft

In the example $T = \text{banana}$, $|\text{occ}(\text{nan})| < |\text{occ}(\text{na})|, |\text{occ}(\text{an})|, |\text{occ}(\text{a})|, |\text{occ}(\text{n})|$, and thus, substring nan is a minimal substring of T . Furthermore, nan is a substring of nana , and $|\text{occ}(\text{nana})| = |\text{occ}(\text{nan})|$. Thus, $\text{cover}(\text{nan}) \subseteq \text{cover}(\text{nana})$ by Lemma 1. (See also Figure 2)

Lemma 1 ensures that if an assignment of variables satisfies the hard clauses C_S for all minimal substrings S of T , then the assignment satisfies the hard clauses C_S for all substrings S of T . With this observation we can conclude that we can omit the hard clauses for all substrings S of T that are not minimal.

The number m of minimal substrings is $O(n)$ because minimal substrings correspond to minimal strings, defined by Blumer et al. [5] based on an equivalence relation over substrings of T , and their number is known to be $O(n)$ (Lemma 3 in [29]). Hence, the total size of the CNF is reduced to $O(mn) \subseteq O(n^2)$.

In particular, the size of the CNF is $o(n^2)$ if $m = o(n)$. We can show that there exists a family of strings $\{T_d\}_{d \in \mathcal{I}}$ for a non-finite set of natural numbers \mathcal{I} with $|T_d| = d^2$ having $o(d^2)$ minimal substrings (hence, for $n = d^2$, $m = o(n)$). To this end, let T_d be the string $S_1 S_2 \cdots S_d$ of length $n = d^2$ over the alphabet $\Sigma = \{\mathbf{a}, \$1, \$2, \dots, \$d\}$, where $S_i = \mathbf{a}^{d-1} \$i$, and \mathbf{a}^{d-1} is the repetition of character \mathbf{a} with length $d - 1$. Then $m = 2\sqrt{n} - 1$ because the minimal substrings of T_d are $\mathbf{a}^1, \mathbf{a}^2, \dots, \mathbf{a}^{\sqrt{n}-1}, \$1, \$2, \dots, \\sqrt{n} .

3.2 Smallest Straight-Line Program as MAX-SAT

To encode a grammar in SAT, we utilize a notion called *grammar parsing* introduced by Rytter [37]. Given an SLP G that produces T , the *parse tree* of T with respect to G is a derivation tree of T , where internal nodes are non-terminal symbols that derive two non-terminal symbols, and leaves are non-terminal symbols that derive a single terminal symbol. The *partial parse tree* of T with respect to G is the tree obtained by pruning the parse tree of T with respect to G so that any internal node is always a first occurrence in a left to right pre-order traversal of the parse tree, i.e., the non-terminal symbol of an internal node is not used in the partial parse tree for any corresponding substring to its left. In other words, if a non-terminal symbol X that derives two non-terminal symbols is a leaf of the partial parse tree, the existence of a unique internal node having the same non-terminal symbol X corresponding to a substring to its left is implied. We will say that the leaf *references* the internal node. The *grammar parsing* of T with respect to G , is the factorization of T consisting of substrings corresponding to the leaves of the partial parse tree of T with respect to G . See Figure 3 for an example.

The *size* of the grammar parsing is equal to the number of leaves in the partial parse tree. It is easy to see that by definition, the internal nodes in the partial parse tree are distinct, consisting of (all) non-terminal symbols that derive two non-terminal symbols. There are σ more non-terminal symbols that derive a single terminal symbol. Therefore, $(\# \text{ of internal nodes}) + \sigma$ is the size of the SLP. Since the partial parse tree is a full binary tree, $(\# \text{ of internal nodes}) = (\# \text{ of leaves}) - 1$, and thus the size of the SLP is equal to $(\text{size of the grammar parsing}) + \sigma - 1$. As σ is independent of the choice of the SLP for T , minimizing the size of the grammar parsing is equivalent to minimizing the SLP.

Our formulation is based on the following lemma.

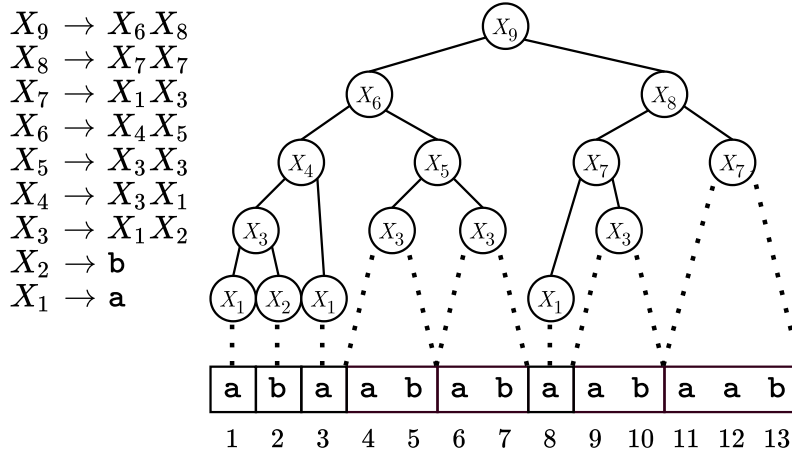


Figure 3 The partial parse tree and the grammar parsing of an SLP for the string $T = \text{abaababaabaab}$. Each internal node is a unique non-terminal symbol. The grammar parsing represented by the rectangles partitioning T is $\text{a, b, a, ab, ab, a, ab, aab}$ of size 8. The size of the SLP is $8 + |\{\text{a, b}\}| - 1 = 9$.

► **Lemma 2.** A factorization $T = F_1 \cdots F_m$ for T is the grammar parsing of an SLP for T if and only if (i) for each factor F_k longer than 1, there exist $i_k < j_k < k$ such that: $F_k = F_{i_k} \cdots F_{j_k}$ and (ii) for any pair of factors $F_x = F_{i_x} \cdots F_{j_x}$ and $F_y = F_{i_y} \cdots F_{j_y}$ longer than 1, (i.e., $(x, y) \in \{(x', y') \mid 1 \leq x', y' \leq m, |F_{x'}| > 1, |F_{y'}| > 1\}$), the intervals $[i_x..j_x]$ and $[i_y..j_y]$ are either disjoint or one is a sub-interval of the other.

Proof. (\Rightarrow) Suppose $F_1 \cdots F_m$ is the grammar parsing of some SLP for T . Then, any F_k longer than 1 has an implied corresponding internal node to the left in the partial parse tree. Since an internal node derives at least two leaves, it derives $F_{i_k} \cdots F_{j_k}$ corresponding to the interval $[i_k..j_k]$ of the factorization for some $i_k < j_k < k$. Furthermore, since all of these intervals are derived from internal nodes of a tree, they must respect the tree structure, i.e., any two of them must be disjoint or contained in one another.

(\Leftarrow) Suppose we are given a factorization $T = F_1 \cdots F_m$ of T , as well as for each F_k , a corresponding interval $[i_k..j_k]$ of the factorization satisfying the conditions of the lemma. Since, for any pair of factors $F_x = F_{i_x} \cdots F_{j_x}$ and $F_y = F_{i_y} \cdots F_{j_y}$, the intervals $[i_x..j_x]$ and $[i_y..j_y]$ are disjoint or contained in one another, we can construct a tree with the internal nodes corresponding to the intervals and the leaves corresponding to the factors of the factorization, where a node is a descendant of another if and only if it is a sub-interval. Although such a tree can be multi-ary in general, we can add internal nodes and transform it into a full binary tree while preserving ancestor/descendant relations of nodes/leaves in the original tree (note that the resulting tree may not be determined uniquely, but its size will always be the same). We assign to each internal node a distinct non-terminal symbol. To each leaf corresponding to a factor F_k longer than 1, we assign the same non-terminal symbol that we assigned to the internal node corresponding to $F_{i_k} \cdots F_{j_k}$. Finally, we assign each leaf corresponding to a factor of length 1 a non-terminal symbol that derives the corresponding terminal symbol. The resulting tree is a partial parse tree for an SLP of size $m + \sigma - 1$ for T with $F_1 \cdots F_m$ as its grammar parsing. ◀

12:8 Computing NP-Hard Repetitiveness Measures via MAX-SAT

We define Boolean variables as follows to encode Lemma 2.

- $f_{i,\ell}$ for $i \in [1, n], \ell \in [1, n + 1 - i]$: $f_{i,\ell} = 1$ if and only if $T[i..i + \ell]$ is a factor of the grammar parsing.
- p_i for $i \in [1, n + 1]$: For $i \neq n + 1$, $p_i = 1$ if and only if i is a starting position of a factor of the grammar parsing. p_{n+1} is for technical reasons. We set $p_1 = p_{n+1} = 1$.
- $ref_{i' \leftarrow i, \ell}$ for $i, \ell, i' \in [1, n]$, s.t. $\ell \geq 2, i' \leq i - \ell$ and $T[i'..i' + \ell] = T[i..i + \ell]$: $ref_{i' \leftarrow i, \ell} = 1$ if and only if $T[i..i + \ell]$ is a factor of the grammar parsing, and the implied internal node of the partial parse tree corresponds to $T[i'..i' + \ell]$.
- $q_{i', \ell}$ for $i' \in [1, n - 1], \ell \in [2, n + 1 - i']$ s.t. $T[i'..i' + \ell]$ has an occurrence in $T[i' + \ell..n]$: $q_{i', \ell} = 1$ if and only if $T[i'..i' + \ell]$ corresponds to an internal node of the partial parse tree that is referenced by at least one factor of the grammar parsing.

We next define constraints that the above variables must satisfy.

First, since each factor of the grammar parsing is disjoint and the concatenation of all factors must be equal to T , the truth values of $f_{i,\ell}$ must uniquely define the truth values for p_i and vice versa. This can be encoded as

$$\forall i \in [1, n], \ell \in [1, n + 1 - i] : f_{i,\ell} \iff p_i \wedge (\neg p_{i+1}) \cdots (\neg p_{i+\ell-1}) \wedge p_{i+\ell} \quad (1)$$

For all i and $\ell \geq 2$ such that $T[i..i + \ell]$ is the first occurrence of a substring $S = T[i..i + \ell]$ of T , $T[i..i + \ell]$ cannot be a factor of a grammar parsing. Thus, we require:

$$\forall i \in [1, n - 1], \ell \in [2, n - i + 1] \text{ s.t. } T[i..i + \ell] \text{ does not occur in } T[1..i] : \neg f_{i,\ell} \quad (2)$$

If $T[i..i + \ell]$ is not the first occurrence of S , $T[i..i + \ell]$ can be a factor. If $T[i..i + \ell]$ is a factor of the grammar parsing of length at least 2, then, there must exist at least one $i' \leq i - \ell$ such that $T[i'..i' + \ell] = S$ and $T[i'..i' + \ell]$ corresponds to an internal node of the partial parse tree. This can be encoded as

$$\begin{aligned} & \forall i \in [1, n], \ell \in [2, n + 1 - i] \text{ s.t. } T[i..i + \ell] \text{ occurs in } T[1..i] : \\ & f_{i,\ell} \implies \bigvee_{i' \in \{k \mid T[k..k + \ell] = T[i..i + \ell], k \in [1, i - \ell]\}} ref_{i' \leftarrow i, \ell}. \end{aligned} \quad (3)$$

Furthermore, for any i, ℓ , a factor $T[i..i + \ell]$ references at most one position, i.e.,

$$\forall i \in [1, n], \ell \in [2, n + 1 - i] : \sum_{i' \in \{k \mid T[k..k + \ell] = T[i..i + \ell], k \in [1, i - \ell]\}} ref_{i' \leftarrow i, \ell} \leq 1. \quad (4)$$

On the other hand, $ref_{i' \leftarrow i, \ell} = 1$ implies that $T[i..i + \ell]$ is a factor of the grammar parsing. Therefore,

$$\begin{aligned} & \forall i \in [1, n], \ell \in [2, n + 1 - i], i' \in \{k \mid T[k..k + \ell] = T[i..i + \ell], k \in [1, i - \ell]\} : \\ & ref_{i' \leftarrow i, \ell} \implies f_{i,\ell} \end{aligned} \quad (5)$$

By definition, it holds that

$$\begin{aligned} & \forall i' \in [1, n - 1], \ell \in [2, n + 1 - i'] \text{ s.t. } T[i'..i' + \ell] \text{ has an occurrence in } T[i' + \ell..n] : \\ & q_{i', \ell} \iff \bigvee_{1 \leq i' + \ell \leq i \leq n} ref_{i' \leftarrow i, \ell} \end{aligned} \quad (6)$$

Next, as shown in Lemma 2, we require that the implied internal node that is referenced by some factor must be an interval of size at least 2 of the factorization. We encode this as:

$$\begin{aligned} & \forall i' \in [1, n-1], \ell \in [2, n+1-i'] \text{ s.t. } T[i'..i'+\ell] \text{ has an occurrence in } T[i'+\ell..n] : \\ & q_{i',\ell} \implies \neg f_{i',\ell} \wedge p_{i'} \wedge p_{i'+\ell} \end{aligned} \quad (7)$$

Also, for any two such implied internal nodes $T[i_1..i_1+\ell_1]$ and $T[i_2..i_2+\ell_2]$, they must either be disjoint, or one is a sub-interval of the other. In other words, it cannot be that a proper prefix interval of one is a proper suffix interval of the other, i.e.,

$$\begin{aligned} & \forall i_1, i_2, \ell_1, \ell_2 \text{ s.t. } i_1 < i_2 < i_1 + \ell_1 < i_2 + \ell_2 \text{ s.t.} \\ & T[i_k..i_k + \ell] \text{ has an occurrence in } T[i_k + \ell..n] \text{ for } k \in \{1, 2\} : \\ & \neg q_{i_1, \ell_1} \vee \neg q_{i_2, \ell_2} \end{aligned} \quad (8)$$

In total, we have $O(n^3)$ Boolean variables dominated by $ref_{i' \leftarrow i, \ell}$. The size of each clause is at most $O(n)$. The total size of the resulting CNF is $O(n^4)$, dominated by Constraint (8) where there are $O(n^4)$ clauses of $O(1)$ size each.

Correctness of the Encoding

We now prove the correctness of our formulation. From Lemma 2, if we are given some SLP producing T , it is clear that the above Boolean variables corresponding to its partial parse tree, referencing structure, and grammar parsing will satisfy all of the constraints.

Next, suppose we are given T and a truth assignment satisfying the above constraints. Starting from the truth assignments of p_i and Constraint (1), we can obtain a factorization of T where we regard $T[i..i+\ell]$ as a factor if and only if $f_{i,\ell} = 1$. For any $i \in [1, n]$ and $\ell \in [2, n-i+1]$, Constraint (2) ensures that $T[i..i+\ell]$ having an occurrence in $T[1..i]$ is a necessary condition for $f_{i,\ell} = 1$. If $f_{i,\ell} = 1$, Constraint (3) implies that there is some $i' \in [1..i-\ell]$ such that $T[i'..i'+\ell] = T[i..i+\ell]$ and $ref_{i' \leftarrow i, \ell} = 1$. From Constraint (4), we know that there is exactly one such i' . On the other hand, Constraint (5) ensures that $ref_{i' \leftarrow i, \ell} = 0$ for all i' when $f_{i,\ell} = 0$. Thus, for each $f_{i,\ell} = 1$ with $\ell > 1$ there exists exactly one i' such that $ref_{i' \leftarrow i, \ell} = 1$, and all other $ref_{\leftarrow \cdot, \cdot}$ are 0. From Constraint (6), it holds that $q_{i',\ell} = 1$ if and only if there is at least one i, ℓ with $ref_{i' \leftarrow i, \ell} = 1$ and thus $f_{i,\ell} = 1$. If $q_{i',\ell} = 1$, from Constraint (7), we have $f_{i',\ell} = 0$, $p_{i'} = p_{i'+\ell} = 1$, implying that $T[i'..i'+\ell]$ is not a factor, but is a concatenation of two or more factors. Constraint (8) requires that all such $T[i'..i'+\ell]$ are either disjoint or that one is a sub-interval of the other.

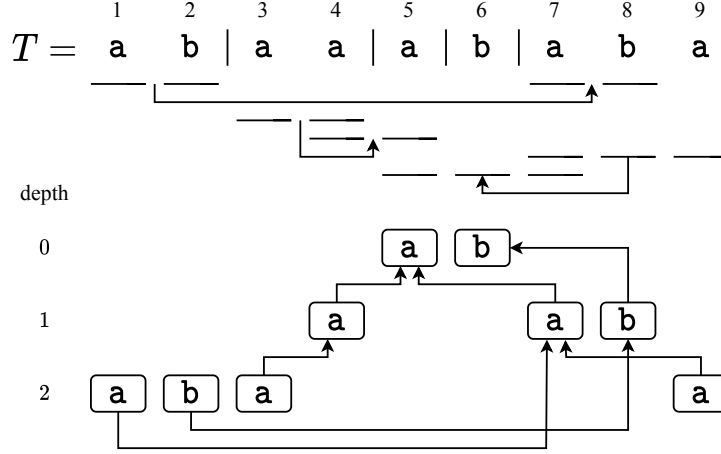
Thus, from the above arguments, we can see that for the factorization defined by the p_i 's, we can associate for each factor, a subinterval of the factorization that satisfies the conditions of Lemma 2, thus implying that the factorization is a grammar parsing of some SLP.

3.3 Smallest Bidirectional Macro Scheme to MAX-SAT

For our SLP encoding, we used the fact that only the leftmost occurrences of the non-terminals are internal nodes – we modeled every later occurrence as a leaf referring to this leftmost occurrence. We could therefore evade the problem of constructing reference cycles since all references point in the same direction. However, in a bidirectional scheme, the references can point in either direction, and the difficulty in defining the encoding is how to ensure that no cycles are introduced in the referencing.

Here, we present a solution that again works with a tree structure, but this time we have multiple trees – a forest that represents the references. In detail, we follow Dinklage et al. [9, Definition 6], who represented a bidirectional macro scheme by a reference forest,

12:10 Computing NP-Hard Repetitiveness Measures via MAX-SAT



■ **Figure 4** Reference forest of the BMS of Figure 1. The forest consists of two trees. The root of each tree corresponds to one of the two ground phrases of the BMS. For instance, to decode $T[1]$, we first need to decode $T[7]$ (the parent of $T[7]$), which has the tree root $T[5]$ as its parent. Hence, the number of ancestors of a node $T[i]$ is the number of references we need to traverse to obtain a ground phrase storing the character of $T[i]$.

where a text position i has text position j as its parent if the phrase covering $T[i]$ has a reference stating that $T[i]$ is copied from $T[j]$. Figure 4 visualizes such a forest. The roots of this reference forest are the positions of the ground phrases.

In order to find a BMS, we go in the inverse direction, and first encode a reference forest from which we subsequently derive a BMS. Since a forest has no cycles, we can use the edges of the forest to define a valid BMS, where each factor has length one (each factor is represented by a node in the reference forest). The final step is to glue together adjacent positions that have adjacent references into larger factors to obtain BMSs with fewer factors.

We start with the encoding for our reference forest. The nodes of the forest coincide with the text positions, and are therefore enumerated from 1 to n . Since a text position i can reference text position j only when $T[i] = T[j]$, it makes sense to restrict j to belong to the set $M_i := \{j \in [1, n] \mid T[i] = T[j], i \neq j\}$. In that case, we say that j is the parent of i . We make use of the following variables.

- $root_i$ for $i \in [1, n]$: $root_i = 1$ if and only if node i is the root of a tree. All roots are at depth 0.
- $dref_{d,i \rightarrow j}$ for $d \in [1, n-1], i \in [1, n], j \in M_i$: $dref_{d,i \rightarrow j} = 1$ if and only if node i at depth d has a parent node j at depth $d-1$.

To obtain a valid reference forest, we define the following constraints. First, each node is a root node or has a parent.

$$\forall i \in [1, n] : root_i + \sum_{d \in [1, n], j \in M_i} dref_{d,i \rightarrow j} = 1 \quad (9)$$

According to Constraint (9), a node i at depth $d \geq 2$ must have exactly one parent j , and j must also have a parent node k (since $d \geq 2$). To enforce acyclicity, we additionally want that k is exactly two levels above of i .

$$\forall d \in [2, n], \forall i \in [1, n], \forall j \in M_i : dref_{d,i \rightarrow j} \implies \sum_{k \in M_j} dref_{d-1, j \rightarrow k} = 1 \quad (10)$$

Next, to translate our reference forest to a BMS, we additionally introduce the following Boolean variables.

- $ref_{i \rightarrow j}$ for $i \in [1, n], j \in M_i$: $ref_{i \rightarrow j} = 1$ if and only if position i references position j .
- p_i for $i \in [1, n]$: $p_i = 1$ if and only if position i is a beginning of a phrase. Note that $p_1 = 1$.

The connection between the variables of the reference forest and our BMS is as follows. For each position $i \in [1, n]$, i can reference at most one position $j \in M_i$, i.e.,

$$\forall i \in [1, n] : \sum_{j \in M_i} ref_{i \rightarrow j} \leq 1 \quad (11)$$

A position i references j if, on any depth d of the reference forest, there is an edge from i to its parent j modeled by $dref_{d,i \rightarrow j}$.

$$\forall d \in [1, n], \forall i \in [1, n], \forall j \in M_i : dref_{d,i \rightarrow j} \implies ref_{i \rightarrow j} \quad (12)$$

Finally, the roots in our reference forest model the ground phrases of the BMS. The roots therefore cannot have a reference, but instead introduce a factor (of length one).

$$\forall i \in [1, n] : root_i \implies p_i. \text{ Additionally, } \forall j \in M_i : root_i \implies \neg ref_{i \rightarrow j} \quad (13)$$

Remembering that the phrases are determined by the variables p_i 's witnessing their starting positions, it is left to model the constraints for the truth assignment of the p_i 's. For that, let us conceptually fix a text position i for which we assume that it references text position j . We consider two cases where $T[i-1]$ and $T[i]$ cannot be in the same phrase. The first case is when i or j are at the start of the text or $j-1 \notin M_{i-1}$:

$$\forall i \in [1, n], j \in M_i \text{ s.t. } i = 1 \text{ or } j = 1 \text{ or } T[i-1] \neq T[j-1] : ref_{i \rightarrow j} \implies p_i \quad (14)$$

The second case is when $j-1 \in M_{i-1}$ but the position $i-1$ does not reference position $j-1$ (it may reference a different position, or it could be a ground phrase):

$$\begin{aligned} \forall i \in [2, n], \forall j \in M_i \text{ s.t. } j > 1 \text{ and } T[i-1] = T[j-1], \\ \neg ref_{i-1 \rightarrow j-1} \wedge ref_{i \rightarrow j} \implies p_i \end{aligned} \quad (15)$$

In total, we have $O(n^3)$ Boolean variables, dominated by $dref_{d,i \rightarrow j}$. The size of the largest clause is $O(n^2)$ due to Constraint (9). The total size of the resulting CNF is $O(n^4)$, dominated by Constraint (10) where there are $O(n^3)$ clauses of $O(n)$ size each.

Correctness of the Encoding

It is easy to see that any valid BMS satisfies the above constraints. We now show that any solution that satisfies the hard clauses yields a valid BMS. The truth assignments for all p_i define a factorization of T . We claim that each position is either a ground phrase, or is assigned exactly one reference consistent with the factorization forming a valid BMS, i.e., the references are acyclic, and, adjacent positions in the same non-ground phrase will refer to adjacent positions thus allowing the phrase to be encoded with the pair of references at both ends of the phrase.

Suppose $p_i = 1$. If $root_i = 1$, then Constraint (9) ensures that all $dref_{\cdot, i \rightarrow \cdot} = 0$ and Constraint (13) ensures that all $ref_{i \rightarrow \cdot} = 0$, i.e., i does not have a reference. Note that, $p_{i+1} = 0$ implies $ref_{i \rightarrow j}$ for some j (shown in the next paragraph), so $p_{i+1} = 1$ must hold. Thus, position i is properly factorized as a ground phrase. If $root_i = 0$, then Constraint (9) ensures that there exist unique d, j such that $dref_{d, i \rightarrow j} = 1$. Furthermore, Constraint (12) ensures that $ref_{i \rightarrow j} = 1$.

Next, consider the case for $p_i = 0$ (which implies $i > 2$). From Constraint (13) we have $root_i = 0$, and from Constraint (15) we have $\forall j > 1 \in M_i$ s.t. $T[i - 1] = T[j - 1]$, $ref_{i \rightarrow j} \implies ref_{i-1 \rightarrow j-1}$. Since $root_i = 0$, Constraint (9) ensures that there exists unique d, j such that $dref_{d, i \rightarrow j} = 1$. Furthermore, Constraint (12) ensures that $ref_{i \rightarrow j} = 1$. Note that due to Constraint (14), neither $j = 1$ nor $T[i - 1] \neq T[j - 1]$ is possible, since this would imply $p_i = 1$, contradicting the assumption that $p_i = 0$. Thus $j > 1$ and $T[i - 1] = T[j - 1]$, and thus we have $ref_{i-1 \rightarrow j-1} = 1$.

The uniqueness of the reference j for each position i of a non-ground phrase is ensured by Constraint (11). Thus, we have that references in adjacent positions in the same non-ground phrase point to adjacent positions. Since the acyclicity of the references are ensured by Constraint (10), we have a valid BMS.

4 Computational Experiments

We have implemented our encodings in PySAT (<https://pysathq.github.io/>) written in the Python language³. As datasets we used the files TRANS, NEWS, E.COLI, and PROGC from the Canterbury and Calgary corpus (<https://corpus.canterbury.ac.nz/>).

Here, we evaluated the sum of the literals in all hard clauses, i.e., the size of the encoded CNF, and the execution time of the SAT solver for computing a solution. In Figure 5, we evaluated our approach on different prefix lengths of the chosen datasets, starting from a prefix of 10 characters up to a prefix with 3000 characters. We aborted an execution after reaching one hour of computation or after exceeding 16 GB of RAM, and hence the lines for computing b and g prematurely end due to these limits on all datasets. Our experiments ran on an Ubuntu 20.04 machine with an AMD Ryzen Threadripper 3990X CPU.

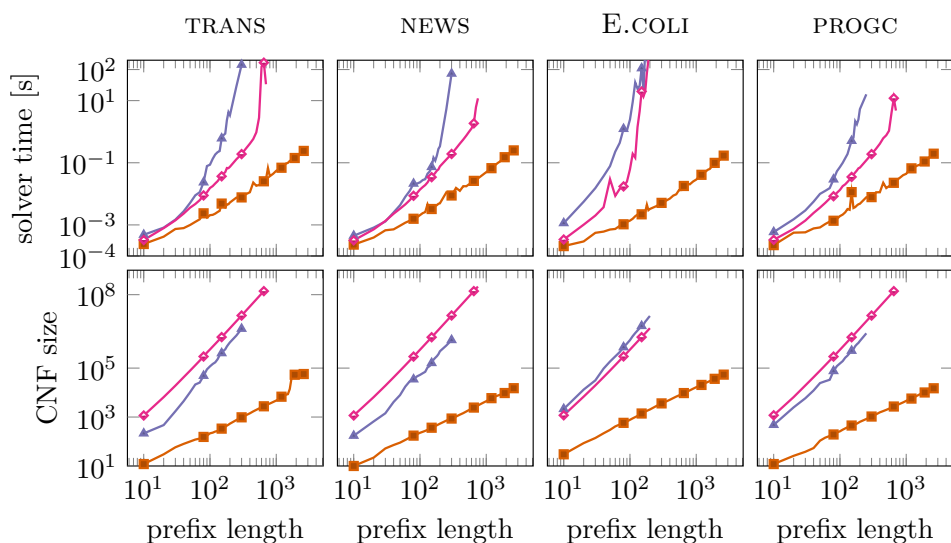
As expected, the size of the encoded CNF correlates with the execution time in all instances. We can see that the encoding for γ needs the least number of literals, and is consequently not only the fastest, but also uses the least amount of memory, allowing us to compute γ for moderately large texts. This is followed by g , and lastly by b . Although the size of the CNF for b is smaller than for g in most cases, clauses formed by Constraint (9) for computing b can become quite large, making the computation cumbersome.

5 Application: Sensitivity of γ

Akagi et al. [1] introduced and studied the notion of *sensitivity* of a repetitiveness measure. Given a repetitiveness measure C (such as γ) for a string T , the sensitivity of C measures how much C can increase when a single character edit operation is performed on T . The authors studied an additive and a multiplicative sensitivity measure. The latter, denoted MS_{op} , is defined as:

$$MS_{op}(C, n) := \max_{T \in \Sigma^n, T' \in \Sigma^*} \left\{ \frac{C(T')}{C(T)} \mid ed_{op}(T, T') = 1 \right\},$$

³ As far as we are aware of, this implementation is single threaded.



■ **Figure 5** Evaluation of our encoded CNFs. The first row shows the running time of PySAT on our CNF instance in seconds. We omit the time needed to specify the CNFs, which is negligible for larger instances. The second row plots the size of the respective CNF. All axes are in logscale.



i.e., the maximum multiplicative increase over all strings with the same length n , where $ed_{op}(T, T') = 1$ means that T' can be built from T by inserting a character into T , or deleting/replacing a character of T . Parameterizing γ with the input string T , for $C(T) = \gamma(T)$, Akagi et al. showed $2 \leq MS_{op}(\gamma, n) \in O(\log n)$.

To improve the lower bound, we conducted exhaustive search for short binary strings when inserting a unique character. This search led us to the string family $\{T_k\}_{k \geq 2}$ with $T_k := \text{abbbaaab}^k$, with which we can improve the lower bound of 2 to $5/2$. For that, let us consider $\gamma(T_k)$ and its size after an insertion of a new character c . First, we observe that $\gamma(T_k) = \gamma(\text{abbbaaab}^k) = 2$. This is because a smallest string attractor is given by $\Gamma(T_2) = \{4, 7\}$ and $\Gamma(T_k) = \{5, 8\}$ for $k \geq 3$ (the characters at the positions in $\Gamma(T_k)$ are underlined). Now let T'_k denote T_k after inserting the character c at text position 9. For $k \geq 5$, it holds that T'_k has a string attractor of size 5, i.e., $\gamma(T'_{k'}) = \gamma(\text{abbbaaabcb}^{k'}) = 5$ for $k' \geq 4$. A minimal string attractor is given by $\Gamma(T'_{k'}) = \{1, 4, 6, 9, 10\}$. We cannot remove a position from $\Gamma(T'_{k'})$ since abb , ba , aab , c , and $\text{b}^{k'}$ are five substrings of $T'_{k'}$, having exactly one occurrence in $T'_{k'}$, and all of them are non-overlapping. Since a string attractor has to be in the cover set of all substrings, we need a string attractor with at least five text positions. Consequently, $MS_{op}(\gamma, n) \geq 2.5$ for any $n \geq 13$ with the insertion or replacement operation.

The availability of computer-aided search facilitated the discovery of strings having certain string attractors.

References

- 1 Tooru Akagi, Mitsuru Funakoshi, and Shunsuke Inenaga. Sensitivity of string compressors and repetitiveness measures. *CoRR*, abs/2107.08615, 2021. [arXiv:2107.08615](https://arxiv.org/abs/2107.08615).
- 2 Hideo Bannai, Mitsuru Funakoshi, Tomohiro I, Dominik Köppl, Takuya Mieno, and Takaaki Nishimoto. A separation of γ and b via Thue-Morse words. In *Proc. SPIRE*, volume 12944, pages 167–178, 2021. [doi:10.1007/978-3-030-86692-1_14](https://doi.org/10.1007/978-3-030-86692-1_14).

- 3 Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.
- 4 Philip Bille, Travis Gagie, Inge Li Gørtz, and Nicola Prezza. A separation between RLSLPs and LZ77. *J. Discrete Algorithms*, 50:36–39, 2018. doi:10.1016/j.jda.2018.09.002.
- 5 Anselm Blumer, J. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987. doi:10.1145/28869.28873.
- 6 Katrin Casel, Henning Fernau, Serge Gaspers, Benjamin Gras, and Markus L. Schmid. On the complexity of the smallest grammar problem over fixed alphabets. *Theory Comput. Syst.*, 65(2):344–409, 2021. doi:10.1007/s00224-020-10013-w.
- 7 Anders Roy Christiansen, Mikko Berggren Ettiienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Trans. Algorithms*, 17(1):8:1–8:39, 2021. doi:10.1145/3426473.
- 8 Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Inf. Process. Lett.*, 106(2):75–80, 2008. doi:10.1016/j.ipl.2007.10.006.
- 9 Patrick Dinklage, Jonas Ellert, Johannes Fischer, Dominik Köppl, and Manuel Penschuck. Bidirectional text compression in external memory. In *Proc. ESA*, pages 41:1–41:16, 2019. doi:10.4230/LIPIcs.ESA.2019.41.
- 10 Patrick Dinklage, Johannes Fischer, Dominik Köppl, Marvin Löbel, and Kunihiko Sadakane. Compression with the tudocomp framework. In *Proc. SEA*, volume 75 of *LIPIcs*, pages 13:1–13:22, 2017.
- 11 Keisuke Goto, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. LZD factorization: Simple and practical online grammar compression with variable-to-fixed encoding. In *Proc. CPM*, volume 9133, pages 219–230, 2015. doi:10.1007/978-3-319-19929-0_19.
- 12 OEIS Foundation Inc. Maximum, over all binary strings w of length n , of the size of the smallest string attractor for w , entry A339391 in the on-line encyclopedia of integer sequences. Accessed: 2022-04-13. URL: <https://oeis.org/A339391>.
- 13 Marek Karpinski, Wojciech Rytter, and Ayumi Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nord. J. Comput.*, 4(2):172–186, 1997.
- 14 Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler transform conjecture. In Sandy Irani, editor, *Proc. FOCS*, pages 1002–1013. IEEE, 2020. doi:10.1109/FOCS46700.2020.00097.
- 15 Dominik Kempa, Alberto Policriti, Nicola Prezza, and Eva Rotenberg. String attractors: Verification and optimization. In *Proc. ESA*, pages 52:1–52:13, 2018. doi:10.4230/LIPIcs.ESA.2018.52.
- 16 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *Proc. STOC*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.
- 17 Dominik Kempa and Barna Saha. An upper bound and linear-space queries on the lz-end parsing. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 2847–2866. SIAM, 2022. doi:10.1137/1.9781611977073.111.
- 18 Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley Professional, 1st edition, 2015.
- 19 Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive measure of repetitiveness. In *Proc. LATIN*, pages 207–219, 2020.
- 20 Sebastian Krefft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theor. Comput. Sci.*, 483:115–133, 2013. doi:10.1016/j.tcs.2012.02.006.
- 21 Kanaru Kutsukake, Takuya Matsumoto, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. On repetitiveness measures of Thue-Morse words. In *Proc. SPIRE*, pages 213–220, 2020. doi:10.1007/978-3-030-59212-7_15.
- 22 N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. In *Proc. DCC*, pages 296–305, 1999. doi:10.1109/DCC.1999.755679.

- 23 Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on information theory*, 22(1):75–81, 1976.
- 24 Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th Edition*. Texts in Computer Science. Springer, 2019. doi:10.1007/978-3-030-11298-1.
- 25 Veli Mäkinen and Gonzalo Navarro. Succinct suffix arrays based on run-length encoding. *Nord. J. Comput.*, 12(1):40–66, 2005.
- 26 Sabrina Mantaci, Antonio Restivo, Giuseppe Romana, Giovanna Rosone, and Marinella Sciortino. A combinatorial view on string attractors. *Theor. Comput. Sci.*, 850:236–248, 2021. doi:10.1016/j.tcs.2020.11.006.
- 27 Sabrina Mantaci, Antonio Restivo, and Marinella Sciortino. Burrows-Wheeler transform and sturmian words. *Inf. Process. Lett.*, 86(5):241–246, 2003. doi:10.1016/S0020-0190(02)00512-4.
- 28 Takuya Mieno, Shunsuke Inenaga, and Takashi Horiyama. Repair grammars are the smallest grammars for fibonacci words. *CoRR*, abs/2202.08447, 2022. arXiv:2202.08447.
- 29 Kazuyuki Narisawa, Hideharu Hiratsuka, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Efficient computation of substring equivalence classes with suffix arrays. *Algorithmica*, 79(2):291–318, 2017. doi:10.1007/s00453-016-0178-z.
- 30 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.
- 31 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.
- 32 Gonzalo Navarro, Carlos Ochoa, and Nicola Prezza. On the approximation ratio of ordered parsings. *IEEE Transactions on Information Theory*, 67(2):1008–1026, 2020.
- 33 Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theor. Comput. Sci.*, 762:41–50, 2019. doi:10.1016/j.tcs.2018.09.007.
- 34 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.
- 35 Takaaki Nishimoto and Yasuo Tabei. LZRR: LZ77 parsing with right reference. *Information and Computation*, page 104859, 2021.
- 36 Luís M. S. Russo, Ana Sofia D. Correia, Gonzalo Navarro, and Alexandre P. Francisco. Approximating optimal bidirectional macro schemes. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2020, Snowbird, UT, USA, March 24-27, 2020*, pages 153–162. IEEE, 2020. doi:10.1109/DCC47342.2020.00023.
- 37 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.
- 38 Hiroshi Sakamoto, Takuya Kida, and Shinichi Shimozone. A space-saving linear-time algorithm for grammar-based compression. In *Proc. SPIRE*, volume 3246, pages 218–229, 2004. doi:10.1007/978-3-540-30213-1_33.
- 39 Hiroshi Sakamoto, Shinichi Shimozone, Ayumi Shinohara, and Masayuki Takeda. On the minimization problem of text compression scheme by a reduced grammar transform. Technical Report 195, Department of Informatics, 2001. URL: https://catalog.lib.kyushu-u.ac.jp/opac_download_md/3045/trcs195.pdf.
- 40 Luke Schaeffer and Jeffrey Shallit. String attractors for automatic sequences. *CoRR*, abs/2012.06840, 2020. arXiv:2012.06840.
- 41 Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005. doi:10.1007/11564751_73.

12:16 Computing NP-Hard Repetitiveness Measures via MAX-SAT

- 42 James A Storer and Thomas G Szymanski. Data compression via textual substitution. *Journal of the ACM (JACM)*, 29(4):928–951, 1982.
- 43 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977.
- 44 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1978. doi:10.1109/TIT.1978.1055934.

Online Metric Allocation and Time-Varying Regularization

Nikhil Bansal ✉

University of Michigan, Ann Arbor, MI, USA

Christian Coester ✉

University of Sheffield, UK

Abstract

We introduce a general online allocation problem that connects several of the most fundamental problems in online optimization. Let M be an n -point metric space. Consider a resource that can be allocated in arbitrary fractions to the points of M . At each time t , a convex monotone cost function $c_t: [0, 1] \rightarrow \mathbb{R}_+$ appears at some point $r_t \in M$. In response, an algorithm may change the allocation of the resource, paying movement cost as determined by the metric and service cost $c_t(x_{r_t})$, where x_{r_t} is the fraction of the resource at r_t at the end of time t . For example, when the cost functions are $c_t(x) = \alpha x$, this is equivalent to randomized MTS, and when the cost functions are $c_t(x) = \infty \cdot \mathbb{1}_{x < 1/k}$, this is equivalent to fractional k -server.

Because of an inherent *scale-freeness* property of the problem, existing techniques for MTS and k -server fail to achieve similar guarantees for metric allocation. To handle this, we consider a generalization of the online multiplicative update method where we decouple the rate at which a variable is updated from its value, resulting in interesting new dynamics. We use this to give an $O(\log n)$ -competitive algorithm for weighted star metrics. We then show how this corresponds to an extension of the online mirror descent framework to a setting where the regularizer is time-varying. Using this perspective, we further refine the guarantees of our algorithm.

We also consider the case of non-convex cost functions. Using a simple ℓ_2^2 -regularizer, we give tight bounds of $\Theta(n)$ on tree metrics, which imply deterministic and randomized competitive ratios of $O(n^2)$ and $O(n \log n)$ respectively on arbitrary metrics.

2012 ACM Subject Classification Theory of computation \rightarrow Online algorithms; Theory of computation \rightarrow K -server algorithms

Keywords and phrases Online algorithms, competitive analysis, k -server, metrical task systems, mirror descent, regularization

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.13

Related Version *Full Version*: <https://arxiv.org/abs/2111.15169>

Funding *Nikhil Bansal*: Supported in part by the NWO VICI grant 639.023.812.

Christian Coester: Supported in part by the Israel Academy of Sciences and Humanities & Council for Higher Education Excellence Fellowship Program for International Postdoctoral Researchers.

Acknowledgements We thank Ravi Kumar, Manish Purohit and Erik Vee for many useful discussions that inspired this work.

1 Introduction

We introduce a natural online problem that generalizes and is closely related to several fundamental and well-studied problems in online computation such as Metrical Task Systems (MTS), the k -server problem and convex body chasing. We call this the *metric allocation* problem (MAP) and it is defined as follows.

There is an underlying metric space M on n points with distances $d(i, j)$ between points i and j . An algorithm maintains an allocation of a resource to the points of M , represented by a vector $x = (x_1, \dots, x_n)$ in the simplex $\Delta = \{x \in \mathbb{R}_+^M \mid \sum_{i \in M} x_i = 1\}$, where x_i denotes the



© Nikhil Bansal and Christian Coester;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, 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

amount of resource at point i . At each time step t , tasks arrive at the points of M . The task at $i \in M$ is specified by a non-increasing and convex cost function $c_{t,i} : [0, 1] \rightarrow \mathbb{R}_+$, which describes the cost of completing the task as function of resource available at i .¹ Given the tasks at time t , the algorithm can modify its previous allocation $x(t-1) \in \Delta$ to $x(t) \in \Delta$. It then incurs a *service cost* $c_t(x(t)) = \sum_{i \in M} c_{t,i}(x_i(t))$ and a *movement cost* of modifying $x(t-1)$ to $x(t)$ according to the distances in M (i.e., sending an ϵ amount of resource from i to j incurs cost $\epsilon \cdot d(i, j)$).

The problem is already very interesting when M is a uniform metric. In particular, this case already goes beyond the reach of existing techniques and highlights a key issue of *scale-freeness* (details in Section 1.2), which seems closely related to current barriers for improving bounds for k -server. For this reason, we will mostly focus on uniform metrics and, more generally, on weighted star metrics. Later, we also describe some results for non-convex cost functions on general metrics.

Besides the connections to other classical problems, that we describe below, MAP also has a natural motivation on its own. For example, the resource may represent workers that can be allocated to various locations. At step t , one could transfer extra workers to locations with high cost to execute tasks more efficiently. This also motivates our assumption on the cost functions being non-increasing (having more resources can only help) and convex (adding extra resources has diminishing returns). If M is a uniform or weighted star metric, this means there is a central depot that workers must return to between switching tasks.

Connections. MAP generalizes several fundamental and well-studied problems in online computation. We describe these next, as they play a key role in our discussion below.

- **Metrical Task Systems.** Here, there is a metric space (M, d) on n points, and the algorithm resides at some point in M at any time. At time t , a cost vector $\alpha_t \in \mathbb{R}_+^M$ arrives. The algorithm can then move from its old location $i_{t-1} \in M$ to a new $i_t \in M$, paying movement cost $d(i_{t-1}, i_t)$ and service cost α_{t,i_t} .

The state of a *randomized* algorithm for MTS is given by a probability distribution $x(t) = (x_1(t), \dots, x_n(t))$ on the n points. Its expected service cost at time t is given by $\sum_i \alpha_{t,i} \cdot x_i(t)$ and its expected movement cost is measured just like in MAP. Thus, randomized MTS is the special case of MAP with cost functions of the form $c_{t,i}(x_i) = \alpha_{t,i} x_i$, i.e., linear and increasing. (We show in the full version that MAP with non-decreasing cost functions is equivalent to MAP with non-increasing cost functions.)

- **k -server.** Here, there is a metric space (M, d) and k servers that reside at points of M . At time t , some point r_t is requested, which must be served by moving a server to r_t . The goal is to minimize the total movement cost. The *fractional* k -server problem is the relaxation² of the randomized k -server problem where points can have a fractional server mass. A request at r_t is served by having a server mass of at least 1 at r_t .

Observe that fractional k -server is the special case of MAP with cost functions $c_{t,i} = 0$ for $i \neq r_t$ and $c_{t,r_t}(x_{r_t}) = \infty$ if $x_{r_t} < 1/k$ and 0 if $x_{r_t} \geq 1/k$, by viewing $kx_i(t)$ as the fractional server mass at location i at time t . Also notice that these $c_{t,i}$ are convex and non-increasing. If M is a uniform metric (or weighted star), this is equivalent to randomized (weighted) paging.

¹ Wlog, we can assume that a task appears at only one point r_t at time t , i.e., $c_{t,i} = 0$ for $i \neq r_t$. See Section 2.

² All known randomized k -server algorithms with poly-logarithmic competitive ratios use this relaxation.

- **Convex function chasing.** Here, the request at time t is a convex function $f_t : \mathbb{R}^n \rightarrow \mathbb{R}_+ \cup \{\infty\}$. The algorithm maintains a point in \mathbb{R}^n , and given f_t it can move from its old position $x(t-1) \in \mathbb{R}^n$ to a new $x(t) \in \mathbb{R}^n$, incurring cost $\|x(t) - x(t-1)\| + f_t(x(t))$.

MAP is a special case of convex function chasing where the norm $\|\cdot\|$ is induced by a metric, cost functions are supported on the unit simplex and have separable form (i.e., $f_t(x) = \sum_{i=1}^n f_{ti}(x_i)$ with f_{ti} monotone).

The convex function chasing problem has seen tremendous progress recently, with the first competitive algorithm for arbitrary dimension given in [14], and $O(n)$ -competitive algorithms shown in [27, 1]; this implies a trivial $O(n)$ upper bound for MAP with convex cost functions.

In recent years there has also been remarkable progress on obtaining polylogarithmic-competitive algorithms for special cases of MAP such as MTS [8, 26, 6, 7, 22, 3, 11, 19, 21] and the k -server problem [20, 4, 5, 2, 12, 25, 16, 24]. However, these solutions are based on rather ad hoc and problem-specific formulations (e.g., considering anti-server mass for k -server vs. server mass for MTS). Due to these inconsistencies, it is also still open, for example, whether for k -server one can achieve the same competitive ratios as for MTS.³

One of our key goals for studying MAP is to develop a systematic and unified approach for understanding a wide class of online problems.⁴

Our contribution. Below we list the results we obtain for MAP, and the new algorithmic design and analysis techniques that we develop. We give more details in Section 1.3.

- We give a tight $O(\log n)$ -competitive algorithm for uniform metrics.
- This result already requires new algorithmic techniques to handle a *scale-freeness* property of the problem. In particular, our algorithm differs from classical multiplicative update algorithms by decoupling the rate at which a variable is updated from its value. The analysis also requires new ideas including a scale-mismatch potential function to handle the differences in the algorithm's perceived scale from the true scale.
- Next, we generalize the $O(\log n)$ bound above to weighted stars, and also refine this guarantee to be $(1 + \epsilon)$ -competitive with respect to the service cost.
- To achieve the refinement, we extend the online mirror descent framework pioneered by Bubeck et al. [12, 11] to a setting with a *time-varying regularizer*. The time-varying nature of the regularizer causes various complications for Bregman divergence based analysis techniques of prior works, and handling them requires several modifications.
- For the generalization of MAP where cost functions can be non-convex, we show an $\Omega(n)$ lower bound for arbitrary metrics. We give a matching $O(n)$ upper bound on tree metrics. This implies an $O(n^2)$ deterministic and $O(n \log n)$ randomized bound on general metrics.
- The $O(n)$ upper bound is also based on the mirror descent framework, but in contrast to all prior works in this framework, we do not use an entropic regularizer, but work with a *simple weighted ℓ_2^2 -regularizer* instead.

³ This contrasts with the deterministic setting, where the competitive ratio of MTS (which is $2n - 1$) is known to be achievable for k -server (where $2k - 1 \leq 2n - 1$ is known). Indeed, this is achieved by the same algorithm for both problems (the work function algorithm; see the book [9] for details) rather than by problem-specific algorithms as in the randomized setting.

⁴ k -server on an n -point metric is also a special case of MTS on a $N = \binom{n}{k}$ point metric. But as the competitive ratio of MTS depends on N , this does not give any interesting bounds for k -server. In contrast, MAP generalizes both fractional k -server and randomized MTS in the *same* metric space.

We next discuss the relevance of allocation problems on star metrics and associated refined guarantees, and then describe the issue of scale-freeness that arises in MAP.

1.1 Allocation problems on star metrics and refined guarantees

Certain special cases of *allocation problems* on star metrics have been studied previously, either implicitly or explicitly, as they capture a lot of the difficulty of *general* metrics. This idea already goes back to Bartal et al. [7]; roughly, one can approximate a general metric space by a hierarchically-separated tree (HST), and recursively run the star algorithm at the internal nodes of the HST to decide how much server mass to allocate to each child subtree. In this way, known algorithms for MTS on general metrics [7, 22, 11, 19] are obtained by using an algorithm for stars as central building blocks.

For k -server, a certain allocation problem on weighted stars was studied in [4] as a first step towards obtaining $\text{polylog}(n, k)$ -competitive algorithms for k -server on general metrics. This allocation problem corresponds to the special case of MAP where the convex functions $c_{t,i}$ are piece-wise linear determined by values $c_{t,i}(j)$ at $j = 0, 1/k, 2/k, \dots, 1$. In subsequent work [2], this step was completed to obtain the first $\text{polylog}(n, k)$ -competitive algorithm for k -server on general metrics.

Refined guarantees. We say that an algorithm has α -competitive service cost and β -competitive movement cost if, up to some fixed additive constant, its service cost is at most α times the *total* (movement plus service) offline cost and its movement cost is at most β times the total offline cost.

For k -server and MTS, polylog-competitive algorithms on general metrics rely on star algorithms with $(1 + \epsilon)$ -competitive service cost and $\text{poly}(\log n, 1/\epsilon)$ -competitive movement cost, for $\epsilon \approx 1/\log n$. The reason is that when the algorithm for stars is used recursively to obtain an algorithm on HSTs, then roughly, the service cost guarantee multiplies across levels and the movement cost guarantee increases additively. See, e.g., [7, 2] for more details.

The algorithm for the special case of MAP considered in [4] has $(1 + \epsilon)$ -competitive service cost and $O(\log k/\epsilon)$ -competitive movement cost. However, the general cost functions that we consider for MAP (in this paper) correspond to this problem as $k \rightarrow \infty$; for this case, the $O(\log k/\epsilon)$ bound of [4] becomes unbounded and does not give anything useful.

1.2 Scale-freeness

The reason for the failure of the algorithm in [4] when $k \rightarrow \infty$ is not just technical, but an inherent one: roughly, for general cost functions, it is unclear how to do *multiplicative updates*, as there is no inherent notion of *scale* in the resulting problem. We elaborate on this issue now, as handling this *scale-freeness* is one of the key conceptual and technical contributions of this paper.

Multiplicative updates. A key underlying idea, sometimes used implicitly, for achieving poly-logarithmic guarantees for k -server, MTS and various other problems (e.g., those based on the online primal dual-framework [18, 17]) is that of multiplicative updates.

Let us see how this works for k -server and MTS on a star metric. For k -server, if a point r is requested, the fractional server amount z_i at other points i is decreased at rate proportional to $1 - z_i + \delta$ (i.e., the amount of server already missing at i plus a small constant δ). On the other hand for MTS, if a cost is incurred at point r , then the other points are increased at rate proportional to $x_i + \delta$.

Multiplicative update for MAP? As MAP generalizes these problems, clearly we also need to do some kind of multiplicative update. However, after some thought one soon realizes that completely unclear is “multiplicative update with respect to what?”.

In particular, if we model k -server as MAP (as described above), then the update rule above becomes $x'_i \propto (1/k - x_i) + \delta$. This is natural as $1/k$ is a fixed parameter with a special meaning as $c_t(x) = \infty$ for $x_{r_t} < 1/k$ and 0 otherwise. On the other hand, for MTS the reason why $x'_i \propto (x_i + \delta)$ is natural is that the $c_{t,i}$ always have x -intercept at 0. In contrast, cost functions in MAP are lacking such an intrinsic *scale*.

We give a more concrete and instructive example to show the difficulty due to this lack of scale.

Example. We saw above how to model k -server via MAP by interpreting $z_i := kx_i$ as the amount of server mass at i , and a request to point i corresponds to the cost function $c_t(x) = \infty \cdot \mathbb{1}_{\{x_i < 1/k\}}$. However, this correspondence between the server mass z_i and the variable x_i is quite arbitrary.

A different way of modeling k -server via MAP is to choose any *offset* vector $a \in [0, 1]^n$ with $s := 1 - \sum_i a_i > 0$, and interpret $z_i := k \cdot (x_i - a_i)/s$ as the server mass at i . Then, a request to page i corresponds to the cost function $c_t(x) = \infty \cdot \mathbb{1}_{\{x_i < a_i + s/k\}}$, and we can additionally intersperse cost functions $c_t(x) = \infty \cdot \mathbb{1}_{\{x_j < a_j\}}$ for each j to ensure that $z_j \geq 0$. In other words, an adversary can simulate a k -server request sequence in various *regions* of the simplex and at different *scales*.

Active region and active scale. Thus, the challenge for an algorithm is to find out the “active region” and “active scale”. Since the adversary can keep changing this region and scale arbitrarily over time, any online algorithm for MAP needs to learn this region dynamically and determine how to do multiplicative updates with respect to the scale and offset of the current region.

At a higher level, the difficulty of learning an “active region” also relates to the difficulty in obtaining a $\text{polylog}(k)$ -competitive algorithm for k -server on general metrics, which seems to require learning a region of $\text{poly}(k)$ many points where the adversary is currently playing its strategy. A step in this direction was made recently in [12].

1.3 Results and techniques

We will first show the following tight bound for uniform metrics.

► **Theorem 1.** *There is an $O(\log n)$ -competitive algorithm for MAP on uniform metrics.*

This bound is the best possible, due to the $\Omega(\log n)$ lower bound for the special case of randomized MTS [10]. Our algorithm is deterministic as randomization does not help for MAP.⁵ The proof of Theorem 1 also extends to weighted stars (and we give a stronger result in Theorem 2 below). However, to introduce the key ideas in a modular way and avoid notational overhead, we focus on uniform metrics first.

To show Theorem 1, a key new idea is to handle the scale-freeness of MAP by decoupling the position x_i and its rate of change x'_i by using separate *rate* variables ρ_i for x'_i . The update of ρ_i is driven by trying to learn the active region and scale (details in Section 3).

⁵ Any randomized algorithm for MAP can be derandomized by tracking its expected location. As cost functions are convex, this can only decrease the algorithm’s cost.

Remark. Theorem 1 has an interesting consequence for the natural case of convex function chasing described above (with separable cost functions supported on the simplex and $\|\cdot\| = \|\cdot\|_1$). The competitive ratio of $O(\log n)$ improves exponentially on the $O(n)$ bound that follows from [1, 27] and breaks the $\Omega(\sqrt{n})$ lower bound that holds for the general case [23, 13].

The following theorem refines the previous guarantee via an improved algorithm for weighted stars.

► **Theorem 2.** *For any $\epsilon > 0$, there exists an algorithm for MAP on weighted stars with $(1 + \epsilon)$ -competitive service cost and $O(\frac{1}{\epsilon} + \log n)$ -competitive movement cost.*

As explained above, a possible application of such refined guarantees is an extension to general metrics.

Time-varying regularization. To achieve the $(1 + \epsilon)$ -competitive service cost, we extend the powerful framework of *regularization* and *online mirror descent* to a setting where the regularizer is *time-varying*. This contrasts with previous works in this framework [15, 12, 11, 16, 19], which all used a *static* regularizer. Also as discussed in Section 1.1, a possible application of such refined guarantees is an extension to general metrics.

A time-varying regularizer is necessary as the regularizer must adapt to the current scale and region over time. This leads to substantial complications in the analysis. In particular, the default potential function for mirror descent analyses – the *Bregman divergence* – is not well-behaved when the offline algorithm moves (actually, it is not even well-defined), and changes of the regularizer lead to uncontrollable changes of the potential. We show how to adapt the Bregman divergence in several ways to obtain a modified potential function that has all the desired properties necessary to carry out the analysis.

Non-convex costs and arbitrary metric spaces. We also consider the version of MAP where the cost functions can be non-convex. Here, the competitive ratio must be exponentially worse.

► **Theorem 3.** *On any n -point metric space, any deterministic algorithm for MAP with non-convex cost functions has competitive ratio at least $\Omega(n)$.*

On tree metrics, we provide a matching upper bound:

► **Theorem 4.** *There is an $O(n)$ -competitive deterministic algorithm for MAP on tree metrics, even if the cost functions are non-convex.*

By known tree embedding techniques, this implies the following result for general metrics:

► **Corollary 5.** *There is an $O(n^2)$ -competitive deterministic and $O(n \log n)$ -competitive randomized algorithm for MAP on arbitrary metric spaces, even for non-convex cost functions.*

ℓ_2^2 -regularization. Our algorithm achieving the tight guarantee on trees is also based on mirror descent, but again with a crucial difference to previous mirror-descent based online algorithms in the literature. While previous algorithms all used some version of an entropic regularizer, our regularizer is a weighted ℓ_2^2 -norm. Here, again, the Bregman divergence is not suitable as a potential function, but the issues are more fundamentally rooted in the non-convex structure of cost functions, and addressing them with changes to the Bregman divergence seems unlikely to work. Instead, our analysis uses two different potential functions, one of which resembles ideas of “weighted depth potentials” used in [12, 11] and the other one is a kind of “one-sided matching”.

1.4 Organization

In Section 2, we define an equivalent version of MAP that will be easier to work with. We will give a first algorithm for uniform metrics in Section 3, where we also describe the ideas to overcome scale-freeness. In Section 4, we discuss a modified algorithm for weighted stars via mirror descent with a time-varying regularizer. However due to space constraints, most of the details are only given in the full version. Our upper and lower bounds for non-convex cost functions on general metrics are proved in the full version, which includes in particular the algorithm based on ℓ_2^2 -regularization.

2 Preliminaries

For $a \in \mathbb{R}$, we write $[a]_+ := \max\{a, 0\}$. A metric space M is called a *weighted star* if there are weights $w_i > 0$ for $i \in M$ and the distance between two points $i \neq j$ is given by $d(i, j) = w_i + w_j$.

Continuous-time model and simplified cost functions. It will be more convenient to work with the following continuous-time version of MAP. Instead of cost functions being revealed at discrete times $t = 1, 2, \dots$, we think of cost functions c_t arriving continuously over time, and that c_t changes only finitely many times. At any time $t \in [0, \infty)$, the algorithm maintains a point $x(t) \in \Delta$, where $\Delta := \{x \in \mathbb{R}_+^M : \sum_{i \in M} x_i = 1\}$, and the dynamics of the algorithm is specified by the derivative $x'(t)$ at each time t . The movement cost and the service cost are given by $\int_0^\infty \|x'(t)\| dt$ and $\int_0^\infty c_t(x(t)) dt$ respectively, where the norm $\|\cdot\|$ is induced by the metric.⁶ On a weighted star metric, the norm $\|\cdot\|$ is given by $\|z\| := \sum_i w_i |z_i|$.

Further, we assume that $c_{t,i}$ is non-zero for only a single location $r_t \in M$ at any time, and $c_{t,i}$ is linear with slope -1 and truncated at 0 (see Figure 1(a)). Formally, $c_t(x) = [s_t - x_{r_t}]_+$ for some $s_t \in [0, 1]$ and $r_t \in M$. A useful consequence of this view is that if the service cost $c_t(x(t))$ incurred by the online algorithm is α_t , then the (one-dimensional) cost function c_{t,r_t} intercepts the x -axis (becomes 0) at the point $x_{r_t}(t) + \alpha_t$. The cost of an offline algorithm at $y \in \Delta$ is then given by $[\alpha_t + x_{r_t}(t) - y_{r_t}]_+$. For a cost function c_t of this form, we will say that $x(t)$ is *charged cost α_t at r_t* .

To simplify the description and analysis of our algorithms, we will further allow them to decrease x_i to a negative value. In the full version, we show that these assumptions are all without loss of generality.

3 A first algorithm for uniform metrics

We describe here an $O(\log n)$ -competitive algorithm for MAP on uniform metrics, proving Theorem 1. Although the algorithm also extends to weighted stars, we assume in this section that all weights are 1 to avoid technicalities and focus on the key ideas.

3.1 Overview

Before we state the formal algorithm, we first give an overview and intuition behind the ideas needed to handle the difficulties due to scale-freeness.

Fix a time t , and let $x = x(t)$ and $y = y(t)$ denote the online and offline position and suppose a cost of $\alpha = \alpha_t$ is received (charged) at point $r = r_t$. We drop t from now for notational ease, as everything is a function of t . Clearly, an algorithm that wishes to be

⁶ In general, $\|z\| = \min_f \sum_{i,j \in M} f(i,j) d(i,j)$, where the minimum is taken over all flows $f: M \times M \rightarrow \mathbb{R}_+$ satisfying $z_i = \sum_{j \in M} (f(j,i) - f(i,j))$ for all $i \in M$.

competitive must necessarily increase x_r (otherwise the offline algorithm can move to some y with $y_r > x_r$ and keep giving such cost functions forever). So, the key question is how to decrease other coordinates x_i for $i \neq r$ to offset the increase of x_r (and maintain $\sum_i x_i = 1$).

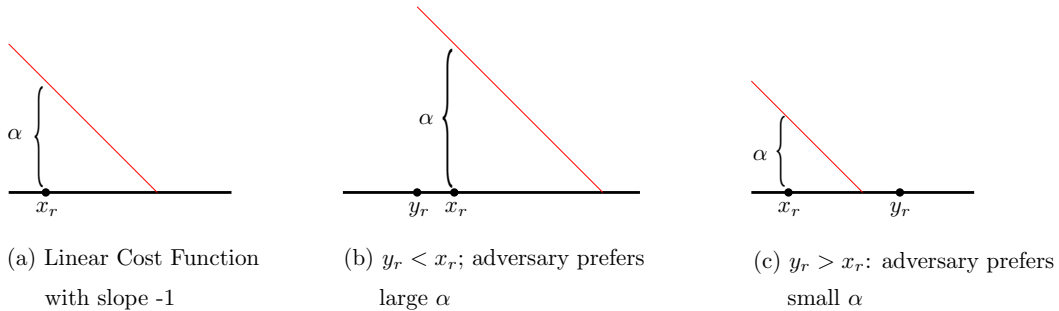
Separate rate variables and how to update them. As discussed in Section 1.2, due to scale-freeness, the rate of update of x_i , denoted x'_i , cannot simply be some function of x_i , as in standard multiplicative update algorithms. So we maintain separate *rate* variables ρ_i (decoupled from x_i) that specify the rate at which to reduce x_i , i.e., $x'_i = -\rho_i$ (plus a small additive term and suitably normalized, but we ignore this technicality for now). Now the key issue becomes how to update these ρ_i variables themselves?

Consider the following two scenarios, which suggest two conflicting updates to ρ_r , depending upon the offline location y_r .

(i) $x_r < y_r$. Here, the adversary can make us incur the service cost while possibly not paying anything itself. However this is not problematic, as we will increase x_r and hence get closer to y_r . Also, decreasing ρ_r is good as it will prevent us in future from decreasing x_r again too fast and move away from y_r when requests arrive at locations $i \neq r$.

(ii) $y_r < x_r$. Here, increasing x_r is fine, as even though we are moving *away* from y_r , the offline algorithm is paying a higher service cost than online. However, decreasing ρ_r is very bad, as this makes it much harder for online to catch up with the offline position y_r later.

To summarize, in case (i), we should decrease ρ_i and in case (ii), we should leave it unchanged or increase it. However, the algorithm does not know the offline location y_r , and hence which option to choose.



■ **Figure 1** Illustration of cost functions.

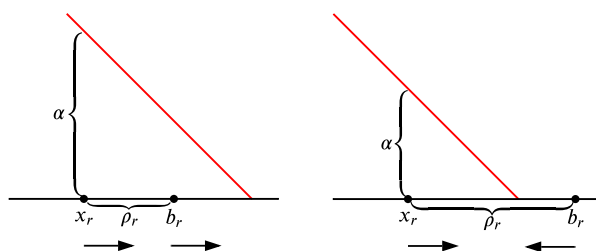
Indirectly estimating y_r . We note that even though the algorithm does not know y_r , it can reasonably estimate whether $y_r < x_r$ or not, by looking at the structure of requests from the adversary’s point of view. Suppose $y_r < x_r$ (see Figure 1(b)). In this case, the adversary will want to give us requests with *large* α ; because for small α , the offline algorithm pays much higher cost in proportion to that of online; indeed, as α gets larger, the ratio between the offline to online service cost tends to 1. On the other hand, if $y_r > x_r$ (see Figure 1(c)), the adversary will tend to give requests so that $x_r + \alpha \leq y_r$ (so offline incurs no service cost at all), and hence keep α *small*.

A problem however is that even though the online algorithm sees α when the request arrives, whether this α is large or small has no intrinsic meaning as this depends on the current scale at which the adversary is giving the instance. The final piece to make this idea work is that the algorithm will also try to learn the scale at which the adversary is playing its strategy. We describe this next.

A region estimate. At each time t , and for each point i , we maintain a real number $b_i \geq 0$ where $b_i \geq x_i$. Intuitively, we can view b_i as an (online) estimate of the “region” where the adversary is playing the strategy, and we set the rate variable $\rho_i = b_i - x_i$. The observations above now suggest the following algorithm. For an incoming request at point r ,

(i) if $x_r + \alpha \leq b_r$ (this corresponds to small α in the discussion above, which suggested that $y_r > x_r$) the algorithm increases x_r and decreases b_r at roughly the same rate (and hence decreases $\rho_r = b_r - x_r$ at roughly twice the rate), and

(ii) if $x_r + \alpha > b_r$ (this corresponds to α being large, which suggested that $y_r < x_r$), the algorithm increases both b_r and x_r at roughly the same rate (and ρ_r only increases slowly). Note that even though the location r is incurring a service cost, we do not necessarily decrease ρ_r .



■ **Figure 2** Illustration of the update rule for ρ_r . If $\alpha > \rho_r$ (left), then x_r and b_r increase and ρ_r changes slowly. If $\alpha < \rho_r$ (right), then x_r and b_r move towards each other and ρ_r decreases.

For other points $i \neq r$, b_i stays fixed. So as x_i decreases, this corresponds to increasing the rate ρ_i . This step is analogous to multiplicative updates, but where the update rate is given by distance of x_i from b_i , where b_i itself might change over time. We also call b_i the “baseline”.

A complication (in the analysis) will be that the b_i themselves are only estimates and could be wrong. E.g., even if b_i is accurate at some given time, the offline algorithm can move y_i somewhere far at the next step, and start issuing requests in that region. The current b_i would be completely off now and the algorithm may make wrong moves. What will help here in the analysis is that the algorithm is quickly trying learn the new b_i .

3.2 Formal description of the algorithm

At any instantaneous time t , the algorithm maintains a point $x(t) \in \Delta$. In addition, it also maintains a point $b(t) \in \mathbb{R}^M$, where $b_i(t) \geq x_i(t)$ for all $i \in M$. Let $\rho_i(t) = b_i(t) - x_i(t)$, for each $i \in M$. We specify the formal algorithm by describing how it updates the points $x(t)$ and $b(t)$ in response to a cost function c_t . Again, we drop t from the notation hereafter.

Suppose the cost function at a given time charges cost α at point r . Then, we increase x_r at rate α and simultaneously decrease all x_i (including x_r) at rate

$$-\alpha \cdot \frac{\rho_i + \delta S}{2S},$$

where $\delta = 1/n$ and $S := \sum_i \rho_i$. Intuitively, one can think of S as the current *scale* of the problem (which is changing over time).

Then the overall update of x can be summarized as

$$x'_i = \alpha \left(\mathbb{1}_{i=r} - \frac{\rho_i + \delta S}{2S} \right) \quad \text{for all } i \in M. \tag{1}$$

13:10 Online Metric Allocation and Time-Varying Regularization

The baseline vector b is updated as follows:

- (i) For $i \neq r$, b_i stays fixed.
- (ii) For $i = r$, if $x_r + \alpha > b_r$ (or equivalently $\alpha > \rho_r$), then $b'_r = \alpha$.
- (iii) For $i = r$, if $x_r + \alpha \leq b_r$ (or equivalently $\alpha \leq \rho_r$), then $b'_r = -\alpha$.⁷

See Figure 2 for an illustration. Notice that the update rules for x'_i and b'_i also ensure that $b_i \geq x_i$. Finally, using that $\rho_i = b_i - x_i$ and writing compactly, this gives the following update rule for ρ_i .

$$\rho'_i = \alpha \left(\frac{\rho_i + \delta S}{2S} - 2 \cdot \mathbb{1}_{i=r \text{ and } \rho_r > \alpha} \right). \quad (2)$$

This completes the description of the algorithm.

Feasibility. Notice that $\sum_i (\rho_i + \delta S)/(2S) = 1$ and hence $\sum_i x'_i = 0$ and the update for x maintains that $\sum_i x_i = 1$. In the algorithm description above, we do not explicitly enforce that $x_i \geq 0$. As we show in the full version, allowing the online algorithm to decrease x_i to negative values is without loss of generality. It is possible to enforce this directly in the algorithm, but this would make the notation more cumbersome.

3.3 Analysis sketch

Due to space constraints, and because our proof of Theorem 2 yields a stronger result anyway, we only provide a brief sketch of the analysis of our algorithm here. It is based on potential functions. Specifically, we define a (bounded) potential Θ that is a function of the online and offline states, and show that at any time t it satisfies

$$\text{On}' + \Theta' \leq O(\log n) \cdot \text{Off}', \quad (3)$$

where On' (resp. Off') denote the change in cost of the online (resp. offline) algorithm, and Θ' is the change in the potential. The potential function Θ consists of two parts defined as

$$\begin{aligned} \text{(Primary potential)} \quad P &:= \sum_{i: x_i \geq y_i} (b_i - y_i) \log \frac{(1 + \delta)(b_i - y_i)}{\rho_i + \delta(b_i - y_i)} \\ \text{(Scale-mismatch potential)} \quad Q &:= \sum_i [\rho_i + 2(x_i - y_i)]_+, \end{aligned}$$

where $y \in \Delta$ is the position of the offline algorithm. The overall potential is given by

$$\Theta = 12P + 6Q.$$

One can verify, by taking derivative with respect to y_i and using $\delta = 1/n$ and $\rho_i = b_i - x_i \geq 0$, that Θ is $O(\log n)$ -Lipschitz in y_i . Thus, the potential increases by at most $O(\log n)$ times the offline movement cost when y changes. It therefore suffices to show (3) for the case that y stays fixed and only the online algorithm moves (i.e., while x , ρ and b are changing).

Recall that $S = \sum_i \rho_i$ is the algorithm's estimate of the current "scale", and define $L := \sum_i |x_i - y_i|$, which we may think of as the true scale of the error between the online position x and the (unknown) offline position y . If the current estimate of the scale is accurate, one would expect $S \approx L$, but in general this need not be true. In the full version, we prove the following two lemmas:

⁷ Strictly speaking, if $x_r + \alpha = b_r$ both rules (ii) and (iii) apply simultaneously and b_r stays fixed.

► **Lemma 6** (Change of the primary potential). *When y is fixed and the online algorithm moves, the change of P is bounded by*

$$P' \leq O(\log n)[\alpha + x_r - y_r]_+ - (\alpha/2) \min\{L/2S, 1\}.$$

► **Lemma 7** (Change of the scale-mismatch potential). *When y is fixed and the online algorithm moves, the change of Q is bounded by*

$$Q' \leq 2\alpha \cdot \mathbb{1}_{y_r \leq x_r + \alpha/2} - (\alpha/2) [1 - L/S]_+.$$

Note that the offline algorithm incurs service cost at rate $[\alpha + x_r - y_r]_+$, and the online algorithm incurs both service and movement cost at rate $O(\alpha)$. In the bound on P' in Lemma 6, the positive term can be charged against the offline service cost. If $S = O(L)$, then the negative term can be used to pay for the online cost. However, if $S \gg L$, then the negative term may be negligibly small. Intuitively, this corresponds to the case that the algorithm's estimate of the scale is far off from the true scale, and this possibility is the reason why we need the scale-mismatch potential.

If $y_r \leq x_r + \alpha/2$, the offline service cost is at least $\alpha/2$, so the positive part in the change of Q is at most 4 times the offline service cost. The negative part in Lemma 7 pays for the online cost if $S \gg L$, which is precisely the case not covered by the primary potential. So,

$$\begin{aligned} \Theta' &= 12P' + 6Q' \leq O(\log n) \cdot \text{Off}' - 3\alpha (\min\{L/S, 2\} + [1 - L/S]_+) \\ &\leq O(\log n) \cdot \text{Off}' - 3\alpha, \end{aligned}$$

which yields the desired inequality (3) because the online service cost is α and online movement cost is at most 2α .

4 Time-varying regularization and refined guarantees on weighted stars

We now turn to an improved algorithm for weighted stars, achieving the refined guarantees of Theorem 2 that the service cost is $(1 + \epsilon)$ -competitive. To do so, we first reinterpret our previous algorithm through the lens of mirror descent.

4.1 Online mirror descent

The online mirror descent framework has been useful to derive optimally competitive algorithms for problems where the state of an algorithm can be described by a point in a convex body (e.g., set cover, k -server, MTS [15, 12, 11, 16, 19]). That is, the algorithm can be described by a path $x: [0, \infty) \rightarrow K$ for a convex body $K \subset \mathbb{R}^n$, such that $x(t)$ describes the state of the algorithm at time t (in our case, $K = \Delta$ is the simplex). In the framework, the dynamics of an algorithm x is specified by a differential equation of the form

$$\nabla^2 \Phi(x(t)) \cdot x'(t) = f(t) - \lambda(t) \tag{4}$$

where $\Phi: K \rightarrow \mathbb{R}$ is a suitable convex function called the *regularizer*, $\nabla^2 \Phi(x(t))$ is its Hessian at $x(t)$, $f: [0, \infty) \rightarrow \mathbb{R}^n$ is called a *control function*, and $\lambda(t)$ is an element of the normal cone of K at $x(t)$, given by

$$N_K(x(t)) := \{\lambda \in \mathbb{R}^n: \langle \lambda, y - x(t) \rangle \leq 0, \forall y \in K\}.$$

Under suitable conditions (which are all satisfied here), the path x is uniquely defined by (4) and absolutely continuous in t [12]. The equation (4) is easiest to read if we imagine that $\nabla^2\Phi(x(t))$ were the identity matrix. Then (4) says that x tries to move in direction $f(t)$, and the normal cone element λ ensures that $x(t)$ does not leave the body K . The case when $\nabla^2\Phi(x(t))$ is different from the identity matrix corresponds to imposing a different (e.g., non-Euclidean) geometry on K . When Φ is fixed, the framework allows a black-box way to prove 1-competitive service cost using the Bregman divergence $D_\Phi(y||x) := \Phi(y) - \Phi(x) + \langle \nabla\Phi(x), x - y \rangle$ as a potential function.

We show in the full version that the dynamics (1) of x defined in the previous section is precisely equivalent to equation (4) when choosing the *time-varying* regularizer

$$\Phi_t(x) := \sum_i (b_i(t) - x_i + \delta S(t)) \log \left(\frac{b_i(t) - x_i}{S(t)} + \delta \right) \quad (5)$$

for $S(t) := \sum_i b_i(t) - 1$, and the control function

$$f(t) := \frac{\alpha_t}{b_{r_t}(t) - x_{r_t}(t) + \delta S(t)} e_{r_t},$$

where e_{r_t} denotes the 0-1-vector with a 1 only in the r_t -coordinate, and α_t is the cost charged at r_t at time t .

To achieve the refined guarantees, we replace α_t by $b_{r_t}(t) - x_{r_t}(t)$ in the control function, and use a very similar regularizer that incorporates a scaling factor and the weights of the star. The most subtle parts in the proof of Theorem 2 are the way the baseline vector b_i is updated (which differs from how it was done in the previous section) and several modifications to the Bregman divergence in order to handle the time-varying nature of Φ_t . Due to space constraints, we defer all details to the full version.

References


- 1 C. J. Argue, Anupam Gupta, Guru Guruganesh, and Ziyi Tang. Chasing convex bodies with linear competitive ratio (invited paper). In *STOC '21*, 2021. doi:10.1145/3406325.3465354.
- 2 Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k -server problem. *J. ACM*, 62(5), 2015. doi:10.1145/2783434.
- 3 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Metrical task systems and the k -server problem on HSTs. In *ICALP' 10*, 2010. doi:10.1007/978-3-642-14165-2_25.
- 4 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Towards the randomized k -server conjecture: A primal-dual approach. In *Symposium on Discrete Algorithms, SODA*, pages 40–55, 2010.
- 5 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012.
- 6 Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS '96*, 1996. doi:10.1109/SFCS.1996.548477.
- 7 Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *STOC '97*, 1997. doi:10.1145/258533.258667.
- 8 Avrim Blum, Howard J. Karloff, Yuval Rabani, and Michael E. Saks. A decomposition theorem for task systems and bounds for randomized server problems. *SIAM J. Comput.*, 30(5), 2000. doi:10.1137/S0097539799351882.
- 9 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
- 10 Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4), 1992. doi:10.1145/146585.146588.

- 11 Sébastien Bubeck, Michael B. Cohen, James R. Lee, and Yin Tat Lee. Metrical task systems on trees via mirror descent and unfair gluing. In *SODA '19*, 2019. doi:10.1137/1.9781611975482.6.
- 12 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k -server via multiscale entropic regularization. In *STOC '18*, 2018. doi:10.1145/3188745.3188798.
- 13 Sébastien Bubeck, Bo'az Klartag, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Chasing nested convex bodies nearly optimally. In *SODA '20*, 2020. doi:10.1137/1.9781611975994.91.
- 14 Sébastien Bubeck, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Competitively chasing convex bodies. In *STOC '19*, 2019. doi:10.1145/3313276.3316314.
- 15 Niv Buchbinder, Shahar Chen, and Joseph (Seffi) Naor. Competitive analysis via regularization. In *Symposium on Discrete Algorithms, SODA*, pages 436–444, 2014.
- 16 Niv Buchbinder, Anupam Gupta, Marco Molinaro, and Joseph (Seffi) Naor. k -servers with a smile: Online algorithms via projections. In *Symposium on Discrete Algorithms, SODA*, pages 98–116, 2019.
- 17 Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2-3):93–263, 2009.
- 18 Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.
- 19 Christian Coester and James R. Lee. Pure entropic regularization for metrical task systems. In *COLT '19*, 2019. URL: <http://proceedings.mlr.press/v99/coester19a.html>.
- 20 Aaron Cote, Adam Meyerson, and Laura J. Poplawski. Randomized k -server on hierarchical binary trees. In *STOC '08*, 2008. doi:10.1145/1374376.1374411.
- 21 Farzam Ebrahimnejad and James R. Lee. Multiscale entropic regularization for MTS on general metric spaces. In *ITCS '22*, 2022.
- 22 Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM J. Comput.*, 32(6), 2003. doi:10.1137/S0097539700376159.
- 23 Joel Friedman and Nathan Linial. On convex body chasing. *Discret. Comput. Geom.*, 9, 1993. doi:10.1007/BF02189324.
- 24 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. A hitting set relaxation for k -server and an extension to time-windows. In *FOCS '21*, 2021.
- 25 James R. Lee. Fusible HSTs and the randomized k -server conjecture. In *FOCS '18*, pages 438–449, 2018. doi:10.1109/FOCS.2018.00049.
- 26 Steve Seiden. Unfair problems and randomized algorithms for metrical task systems. *Inf. Comput.*, 148(2), 1999. doi:10.1006/inco.1998.2744.
- 27 Mark Sellke. Chasing convex bodies optimally. In Shuchi Chawla, editor, *SODA '20*, 2020. doi:10.1137/1.9781611975994.92.

An Upper Bound on the Number of Extreme Shortest Paths in Arbitrary Dimensions

Florian Barth  

Universität Stuttgart, Germany

Stefan Funke 

Universität Stuttgart, Germany

Claudius Proissl 

Universität Stuttgart, Germany

Abstract

Graphs with multiple edge costs arise naturally in the route planning domain when apart from travel time other criteria like fuel consumption or positive height difference are also objectives to be minimized. In such a scenario, this paper investigates the number of *extreme shortest paths* between a given source-target pair s, t . We show that for a fixed but arbitrary number of cost types $d \geq 1$ the number of extreme shortest paths is in $n^{O(\log^{d-1} n)}$ in graphs G with n nodes. This is a generalization of known upper bounds for $d = 2$ and $d = 3$.

2012 ACM Subject Classification Theory of computation \rightarrow Shortest paths

Keywords and phrases Parametric Shortest Paths, Extreme Shortest Paths

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.14

Funding This work was in part supported by the Deutsche Forschungsgemeinschaft (DFG) within the priority program 1894: Volunteered Geographic Information: Interpretation, Visualization and Social Computing.

Acknowledgements We thank Kshitij Gajjar and Jaikumar Radhakrishnan for their very helpful input and for identifying an error in a previous version of this paper.

1 Introduction

Given a finite set of points $P \subset \mathbb{R}^d$, what is the number of vertices (or extreme points) of the convex hull of P ? This is a frequently asked question, for instance, in the area of Multi-objective Linear Programming [3] or in probability theory [4]. In this work, we examine this question for the case when P is the set of cost vectors of paths in a graph $G(V, E)$ with multiple edge costs.

Multi-objective path computation has obvious applications in the transportation domain, where the cost values (also called metrics) might correspond to quantities like travel time, fuel consumption, or positive height difference.

Figure 1 shows example cost vectors with two metrics. The red points are non-dominated (or Pareto-optimal) and, thus, may be the solution to constrained minimization problems. However, optimizing over all non-dominated cost vectors often turns out to be too expensive as, for instance, in the constrained shortest path problem [12, 14]. A typical strategy in such cases is to restrict the set of possible solutions to the non-dominated extreme points of the convex hull (circled in blue). The extreme points have the property that for any convex combination of the metrics there is at least one extreme point optimal for it. Therefore, extreme points are interesting on their own and one can hope that restricting the search to extreme points will lead to a good approximation of the optimal solution.



© Florian Barth, Stefan Funke, and Claudius Proissl;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

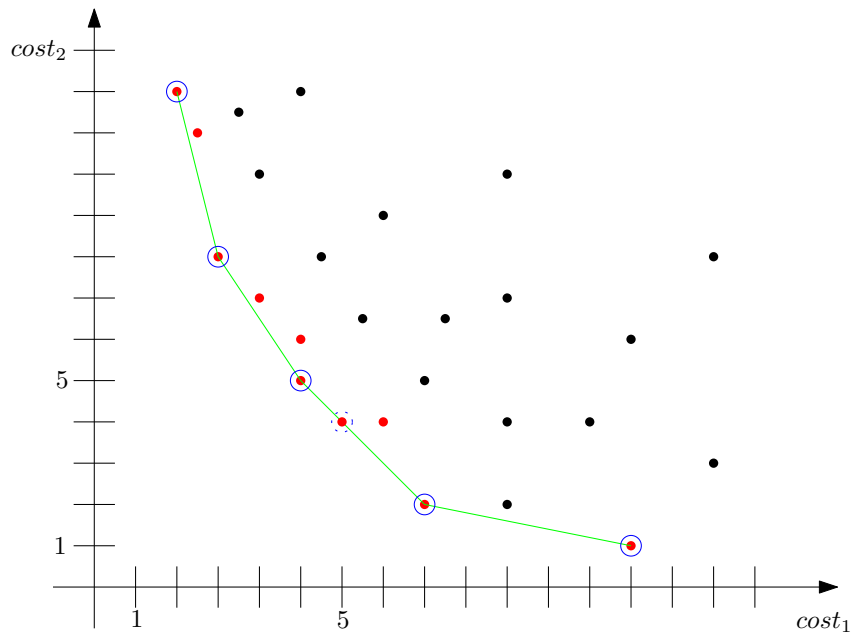
Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 14; pp. 14:1–14:12
Leibniz International Proceedings in Informatics



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

Let P_{st} be the set of cost vectors of all simple paths from a node s to a node t in a graph G . While it is easy to see that the number of non-dominated points in P_{st} can be exponential in the size of G , there is little known about the complexity of the extreme points in P_{st} . In this work we tackle the problem of counting the extreme points in P_{st} , which we call *extreme shortest paths*. We show that the number of extreme shortest paths in P_{st} is in $n^{O(\log^{d-1} n)}$, where n is the number of nodes in G and d is the fixed number of metrics. Thus, complexity-wise there is indeed a considerable gap between extreme points and non-dominated points in P_{st} .

There are multiple ways to model multi-objective shortest path problems. A well established one is the parametric shortest path problem, which is typically formulated for the two-metric case. We have decided to use the variant developed in the context of personalized route planning [10, 8] as it fits very well to practical application scenarios. Complexity-wise there is no difference between these two models.



■ **Figure 1** Paths in cost space (black and red dots); Pareto-optimal paths (red); (lower left part of) the boundary of the convex hull of all Pareto-optimal paths in green; extreme shortest paths/extreme points of the CH circled in blue; shortest (but not extreme) path dot-circled in blue.

Related Work

For the comparison with related work let us first define the well-studied parametric shortest path problem as in [9]. Given an acyclic, directed graph $G(V, E)$ the weights w_e of the edges $e \in E$ are linear functions of the form

$$w_e(\lambda) := a_e \lambda + b_e.$$

The cost of a path p is then given by $C(p, \lambda) := \sum_{e \in p} w_e(\lambda)$. Clearly, if we compute the shortest path from a node $s \in V$ to a node $t \in V$ for different values of λ , we may get different paths. Let Π_{st} bet the set of all paths from s to t . Then

$$C(\lambda) := \min_{p \in \Pi_{st}} C(p, \lambda)$$

is the shortest path cost function, which is a concave, piece-wise linear function. The maximum possible number of pieces of $C(\lambda)$ with respect to the size of G is called the *parametric shortest path complexity*. Complexity-wise, counting the pieces of $C(\lambda)$ is equivalent to counting extreme shortest paths from s to t . Therefore, all results regarding the parametric shortest path complexity are closely related to our work.

For the two-metric case, it is well known that the number of pieces in $C(\lambda)$ is upper bounded by $n^{O(\log n)}$, where n is the number of nodes in G [11]. This upper bound is tight [5, 6, 15], even for planar graphs [9].

Gajjar and Radhakrishnan [9] extend the parametric shortest path problem to three dimensions by setting

$$w_e(\lambda := (\lambda_1, \lambda_2, \lambda_3)) := a_e \lambda_1 + b_e \lambda_2 + c_e \lambda_3.$$

They show that in this case the number of extreme shortest paths is in $n^{(\log n)^2 + O(\log n)}$.

We are not aware of any results regarding the parametric shortest path complexity beyond three dimensions. Parts of our way (especially Section 3.2) to prove the general upper bound are inspired by the proof for three dimensions in [9].

Both, Pareto-optimal paths as well as extreme shortest paths have been instrumented to create alternative route recommendations. The former approach, pursued e.g. in [7, 13], unfortunately only seems to be viable on rather small graphs due to the too rapidly growing number of Pareto-optimal paths. Restricting to extreme shortest paths, though, as in [8], has been shown to be feasible in different practical application scenarios [2, 1].

2 Preliminaries

In this section, we introduce the notions used in Section 3 and show some basic, well known properties.

For a set $f \subseteq \mathbb{R}^d$, we define its dimension $\dim(f)$ to be the maximum number of affinely independent points in f minus one. For a finite set $P \subset \mathbb{R}^d$ we denote the number of elements in P with $|P|$.

► **Definition 1.** *The d -metric preference space \mathcal{P}_d is defined as follows.*

$$\mathcal{P}_d := \{(\alpha_1, \alpha_2, \dots, \alpha_d) \in \mathbb{R}_{\geq 0}^d \mid \sum_{i=1}^d \alpha_i = 1\}$$

Note that the d -metric preference space is a $(d-1)$ -dimensional simplex in d dimensions. Given a finite set of points $P \subset \mathbb{R}^d$ with $v \in P$, let $f_P(v) \subseteq \mathcal{P}_d$ be the set of preferences for which αv^T is minimal, where αv^T is the dot product of the (row) vectors α, v . Or more formally,

$$f_P(v) := \{\alpha \in \mathcal{P}_d \mid \alpha(v - v')^T \leq 0 \forall v' \in P\}. \quad (1)$$

► **Definition 2.** *Given a finite set of points $P \subset \mathbb{R}^d$, a subset $P' \subseteq P$ is a preference cover (PC) of P if and only if*

$$\bigcup_{v \in P'} f_P(v) = \mathcal{P}_d. \quad (2)$$

The following lemma states that there is a special, minimal PC for each finite point set P .

14:4 An Upper Bound on the Number of Extreme Shortest Paths in Arbitrary Dimensions

► **Lemma 3.** *Given a finite set of points $P \subset \mathbb{R}^d$ and let $X := \{v \in P \mid \dim(f_P(v)) = d-1\}$, then X is a PC of P and it holds*

$$|X| = \min_{P' \text{ is PC of } P} |P'|.$$

Proof. Let $\mathcal{M} := \min_{P' \text{ is PC of } P} |P'|$. As the number of points in P is finite and because the inequality in (1) is not strict, it holds

$$\bigcup_{v \in X} f_P(v) = \mathcal{P}_d.$$

Thus, X is a PC of P and $|X| \geq \mathcal{M}$. Moreover, for each $v \in X$ one can find an $\alpha \in \mathcal{P}_d$ with

$$\alpha v^T < \alpha v'^T \quad \forall v' \in P \setminus \{v\}.$$

It follows that $|X| \leq \mathcal{M}$ and, thus, $|X| = \mathcal{M}$. ◀

The following definition is motivated by Lemma 3.

► **Definition 4.** *Given a finite set of points $P \subset \mathbb{R}^d$, we call the subset*

$$\mathcal{M}(P) := \{v \in P \mid \dim(f_P(v)) = d-1\}$$

minimum preference cover (MPC) of P .

► **Definition 5.** *Given a finite set of points $P \subset \mathbb{R}^d$, the d -metric preference space subdivision (PSS) $\mathcal{S}_d(P)$ (or simply $\mathcal{S}(P)$) is the arrangement induced by the set $\{f_P(v) \mid v \in P\}$. We write $f_P^r \in \mathcal{S}_d(P)$ for an r -dimensional facet of $\mathcal{S}_d(P)$.*

Figure 2 shows an example 3-metric PSS. The following lemma motivates the term *subdivision*.

► **Lemma 6.** *Given a finite set of points $P \subset \mathbb{R}^d$ with $d > 1$, for any two $v, v' \in \mathcal{M}(P)$ we have $\dim(f_P(v) \cap f_P(v')) < d-1$. Furthermore, for any $v \in P$ there is a $v' \in \mathcal{M}(P)$ with $f_P(v) \subseteq f_P(v')$.*

Proof. Given two points $v, v' \in \mathcal{M}(P)$, let H be the hyperplane described by $\alpha(v-v')^T = 0$, $\alpha \in \mathbb{R}^d$. For the first part of the lemma we have to show that $\dim(H \cap \mathcal{P}_d) \leq d-2$. This is the case if $\mathcal{P}_d \not\subseteq H$. The vector $v-v'$ must have at least one positive and one negative entry (otherwise, one vector would dominate the other). Thus, H cannot be parallel to $\sum_{i \leq d} \alpha_i = 1$ and $\mathcal{P}_d \not\subseteq H$ follows.

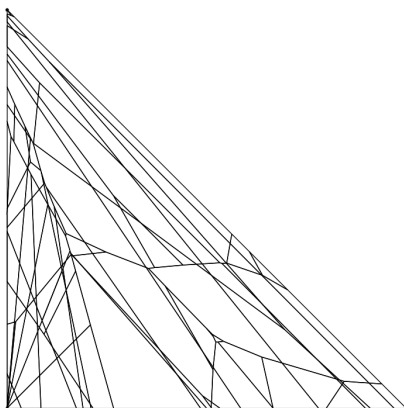
Now we come to the second part of the lemma. If $\dim(f_P(v)) = d-1$, then $v \in \mathcal{M}(P)$ and we are finished. Thus, we may assume that $0 < \dim(f_P(v)) < d-1$. From Lemma 3 we know that

$$f_P(v) = \{\alpha \in \mathbb{R}^d \mid \alpha(v-v')^T \leq 0 \quad \forall v' \in \mathcal{M}(P)\}.$$

Therefore, if $\dim(f_P(v)) < d-1$, there must be a point $v' \in \mathcal{M}(P)$ and a hyperplane H' described by $\alpha(v-v')^T = 0$ with $f_P(v) \subseteq H'$. It follows that $f_P(v) \subseteq f_P(v')$. ◀

In fact, a PSS looks similar to a Voronoi diagram.

► **Definition 7.** *Given a finite set of points $P \subset \mathbb{R}^d$, then $\varphi_d^r(P)$ is the number of r -dimensional facets in the PSS $\mathcal{S}(P)$.*



■ **Figure 2** Example 3-metric preference space subdivision \mathcal{S}_3 .

Throughout this work, we assume that a graph $G(V, E)$ with directed edges, without multi-edges and with n nodes is given. A path $\pi := u_1 e_1 u_2 e_2 \dots e_l u_{l+1}$ in G is an alternating sequence of nodes and edges that starts and ends with a node. Furthermore, for each edge $e_i \in \pi$ it holds $e_i = (u_i, u_{i+1})$. We say that a path π is simple if it contains each node $u \in V$ at most once.

A d -metric (or d -dimensional) cost function $c : E \rightarrow \mathbb{R}_{\geq 0}^d$ maps edges to (non-negative) cost vectors. We extend c to paths π as follows.

$$c(\pi) := \sum_{e \in \pi} c(e)$$

We define \mathcal{C}_d to be the set of all d -metric cost functions of E .

Let Π be a set of paths (or: path set) in G and c a cost function for E . Then we define the point set $P(\Pi, c)$ as follows.

$$P(\Pi, c) := \{c(\pi) \mid \pi \in \Pi\}$$

With $\Pi_{st}(l)$ we denote all simple paths from $s \in V$ to $t \in V$ (with s and t being arbitrary but fixed nodes) with at most $\lceil l \rceil$ edges (l is a real number as we need to divide it by two in a recursion later on). We call $\Pi_{st}(l)$ a *complete path set*.

► **Definition 8.** Given a path set Π , we define $\varphi_d^r(\Pi)$ with $1 \leq r \leq d-1$ as

$$\varphi_d^r(\Pi) := \max_{c \in \mathcal{C}_d} \varphi_d^r(P(\Pi, c)).$$

Note that, by Lemma 6, it holds $\varphi_d^{d-1}(\Pi) = \max_{c \in \mathcal{C}_d} |\mathcal{M}(P(\Pi, c))|$.

► **Definition 9.** Given a path set Π and a cost function $c \in \mathcal{C}_d$, an element $v \in P(\Pi, c) =: P$ is an extreme shortest path with respect to Π and c if and only if $v \in \mathcal{M}(P)$. Furthermore, for a preference $\alpha \in \mathcal{P}_d$ we call a cost vector $v \in \mathcal{M}(P)$ α -shortest path with respect to P if and only if $\alpha \in f_P(v)$.

There are a few possibly confusing things to clarify here. First, we call the elements in $\mathcal{M}(P)$ extreme shortest paths even though they are mere cost vectors. The reason is that there is a bijective relationship between elements in P and Π as long as no two paths in Π have the

same cost vector. However, if there are multiple paths in Π with the same cost vector we prefer to count them only once. This is the reason why we focus on the set P instead of the set Π .

Second, if a cost vector v is an extreme shortest path depends on the path set Π . For instance, if $\Pi := \{\pi\}$, then any cost vector $v = c(\pi)$ is an extreme shortest path. Thus, if we say that v is an extreme shortest path, this is always with respect to some path set Π and cost function c that we either mention explicitly or that should be clear from the context.

Third, a cell of a d -metric PSS $\mathcal{S}_d(P)$ is a $(d - 1)$ -dimensional and not a d -dimensional facet. The reason is that the preference space \mathcal{P}_d lives in d dimensions but is itself a $(d - 1)$ -dimensional object (because the preferences sum up to one).

This work is about finding an upper bound for the number of extreme shortest paths in $P(\Pi_{st}(n), c)$ in arbitrary dimensions d and for arbitrary cost functions $c \in \mathcal{C}_d$.

Comparison of Personalized Route Planning and Parametric Shortest Path Problem

In the parametric shortest path problem, as defined for three dimensions in [9], the edge costs have the form

$$c(e \in E, \lambda \in \mathbb{R}^d) := \lambda c(e)^T \tag{3}$$

with $c : E \rightarrow \mathbb{R}^d$. In contrast to the definition in [9], λ_1 is typically set to one. However, note that the shortest path problem is homogeneous in the sense that if we scale all edge costs by a factor $\delta > 0$ we also scale the extreme shortest paths by δ . Thus, as argued in [9], (3) can always be scaled to either $\lambda_1 = 1$ or $\lambda_1 = -1$. The complexity of these two versions therefore can only differ by a factor of two.

In the personalized route planning model the edge costs are defined as

$$c(e \in E, \alpha \in \mathcal{P}_d) := \alpha c(e)^T \tag{4}$$

with $c : E \rightarrow \mathbb{R}_{\geq 0}^d$. The differences between (4) and (3) are that the preferences $\alpha \in \mathcal{P}_d$ and cost vectors are non-negative and the preferences sum up to one. We can handle the normalization with the same scaling argument as above. The non-negativity makes sure that there are no negative cost cycles in G regardless of α . This issue is treated differently in the parametric shortest path problem by requiring G to be acyclic. In the end, it is a matter of taste which requirement to choose. With minor adjustments our proofs also work for the case when G is acyclic and the costs and preferences are allowed to be negative.

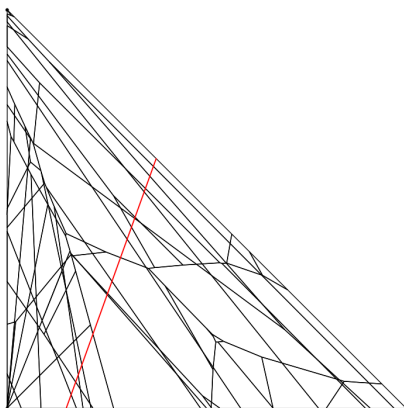
3 A General Upper Bound on the Number of Extreme Shortest Paths

In this section we prove the following theorem.

► **Theorem 10.** *For any fixed but arbitrary $d \geq 1$ and any cost function $c \in \mathcal{C}_d$ the number of extreme shortest paths in $P(\Pi_{st}(n), c)$ is upper bounded by $n^{O(\log^{d-1} n)}$.*

► **Definition 11.** *We define $\varphi_d^r(n, l)$ to be the maximum of $\varphi_d^r(\Pi_{st}(l))$ over all possible graphs $G(V, E)$ with n nodes and all possible node pairs $s, t \in V$.*

Thus, Theorem 10 is equivalent to the statement $\varphi_d^{d-1}(n, n) \in n^{O(\log^{d-1} n)}$. We prove Theorem 10 with a recursion of the form $\varphi_d^{d-1}(n, n) \leq f(\varphi_{d-1}^{d-2}(n, n))$. Thus, we upper bound $\varphi_d^{d-1}(n, n)$ with a recursion in the dimension d , which is an idea also used in the proof of the upper bound for $d = 3$ in [9].



■ **Figure 3** Example preference space subdivision with red hyperplane intersection.

We obtain the function f as follows. First, we show that the intersection of a d -dimensional PSS and a hyperplane can have at most the complexity of a $(d - 1)$ -dimensional PSS (Lemma 12). In fact, this observation is based on a similar result shown in [9] in the context of parametric shortest paths. Second, we use this insight to upper bound $\varphi_d^{d-2}(n, l)$ for special path sets (Lemma 13 and 14), which then allows us to construct a second recursion of the form $\varphi_d^{d-2}(n, l) \leq g(\varphi_d^{d-2}(n, \frac{l}{2}))$. This leads to an upper bound for $\varphi_d^{d-2}(n, l)$ based on $\varphi_{d-1}^{d-2}(n, l)$ (Lemma 15). The proof of Theorem 10 then uses Lemma 15 together with an observation we discuss in Section 3.1 to construct the function f . We consider Lemma 13 and 14 in Section 3.3 as our main contributions as these ingredients allow us to generalize the ideas shown in [9] to arbitrary values d .

3.1 A First Upper Bound on the Number of Cells

The number of $(d - 2)$ -dimensional facets in a PSS is an upper bound of the number of $(d - 1)$ -dimensional facets in the same PSS for the following reasons. Given any finite point set $P \subset \mathbb{R}^d$ with $d > 1$. Each facet f_P^{d-2} of $\mathcal{S}_d(P)$ supports at most two $(d - 1)$ -dimensional facets. Moreover, every facet f_P^{d-1} is supported by at least d $(d - 2)$ -dimensional facets. This is true because the preference space \mathcal{P}_d is bounded itself and, thus, there is no unbounded cell in $\mathcal{S}_d(P)$. Therefore, we have

$$\varphi_d^{d-1}(P) \leq \frac{2}{d} \varphi_d^{d-2}(P). \quad (5)$$

For the case $d = 1$, our task of counting extreme shortest paths is simple. It holds

$$\varphi_1^0(P) \leq 1 \quad (6)$$

for any finite point set P because $\mathcal{M}(P) = \{\min_{v \in P} v\}$.

3.2 Bounding the Complexity of PSS Intersections

In this section we show that the complexity of the intersection of a hyperplane with a d -metric PSS is upper bounded by the complexity of a $(d - 1)$ -metric PSS. This observation is a crucial ingredient of our proof of Theorem 10 as it allows us to construct recursive upper bounds in the dimension d . The authors of [9] prove a similar statement in the context of parametric shortest paths. As their setting and notation slightly differ from ours, we decided to give a proof of the following lemma.

► **Lemma 12.** *Let $d > 1$ and let H_d be a hyperplane in d dimensions with $\mathcal{P}_d \not\subseteq H_d$ that intersects the preference space \mathcal{P}_d . Then for any path set Π and cost function $c \in \mathcal{C}_d$ the set $Y := \{f_{P(\Pi, c)}^{d-1} \cap H_d \mid \dim(f_{P(\Pi, c)}^{d-1} \cap H_d) = d - 2\}$ has at most $\varphi_{d-1}^{d-2}(\Pi)$ elements.*

Proof. We fix an arbitrary cost function $c \in \mathcal{C}_d$ and define $P := P(\Pi, c)$. See Figure 3 for an example of the intersection $H_d \cap \mathcal{P}_d$. The set $Z := \{f \cap H_d \mid f \in \mathcal{S}_d(P)\}$ looks like a $(d - 1)$ -metric PSS. However, it is unclear whether the complexity bounds for $(d - 1)$ -metric PSS also apply for sets like Z , which live in d dimensions. This lemma says that the answer is yes in the case of extreme shortest paths.

Our strategy to prove Lemma 12 looks as follows.

1. We show that for each $v \in P$ there is at most one $f \in Y$ with $\dim(f_P(v) \cap f) = d - 2$.
2. We define mappings $\eta : \mathbb{R}^d \rightarrow \mathbb{R}^{d-1}$ and $\beta : H_d \cap \mathcal{P}_d \rightarrow \mathcal{P}_{d-1}$ such that for each $\alpha \in H_d \cap \mathcal{P}_d$ and each $v \in P$ it holds $\alpha v^T = \beta(\alpha) \eta(v)^T$.

We prove these points later on and first show that they are sufficient to prove the lemma. Let $\eta(P)$ and $\beta(f_P)$ be the mapped sets P and f_P . It follows from the second point that for any $v \in P$ with $\dim(f_P(v) \cap H_d) = d - 2$ it holds

$$\beta(f_P(v) \cap H_d) \subseteq f_{\eta(P)}(\eta(v)).$$

Therefore, for each $f \in Y$ there is also an extreme shortest path $v' \in \mathcal{M}(\eta(P))$ with $\beta(f_P(v)) \subseteq f_{\eta(P)}(\eta(v'))$ (see Lemma 6).

Let $v' \in \mathcal{M}(\eta(P))$ be any extreme shortest path with respect to $\eta(P)$. Then the set $W := \{f \in Y \mid \beta(f) \subseteq f_{\eta(P)}(v')\}$ contains at most one element. Otherwise, we could take any cost vector $v \in P$ with $\eta(v) = v'$ and show, using point two, that all elements of W are subsets of $f_P(v)$. This contradicts the first point.

In summary, for each element $f \in Y$ we find an extreme shortest path $v' \in \mathcal{M}(\eta(P))$ with $\beta(f) \subseteq f_{\eta(P)}(v')$. Furthermore, for each extreme shortest path $v' \in \mathcal{M}(\eta(P))$ we find at most one $f \in Y$ with $\beta(f) \subseteq f_{\eta(P)}(v')$. Thus, it is possible to injectively map Y to $\mathcal{M}(\eta(P))$. As $|\mathcal{M}(\eta(P))| \leq \varphi_{d-1}^{d-2}(\Pi)$ by definition and because we chose the cost function c arbitrarily, this finishes the proof.

It remains to show that the two points are correct. Assume that we find a cost vector $v \in P$ and two elements $f_1, f_2 \in Y$ with $f_1 \subseteq f_P(v)$ and $f_2 \subseteq f_P(v)$. We know from Lemma 6 that there is a $v' \in \mathcal{M}(P)$ with $f_P(v) \subseteq f_P(v')$. Let $f_3 := f_P(v') \cap H_d$. Clearly, $f_1 \subseteq f_3$ and $f_2 \subseteq f_3$. As $f_1 \neq f_2$ at least one of them cannot be equal to f_3 . W.l.o.g. let $f_2 \subset f_3$.

By definition of Y , we find a shortest path $v_2 \in \mathcal{M}(P)$ with $f_2 = f_P(v_2) \cap H_d$. As $f_2 \subset H_d$ and $f_3 \subset H_d$ it follows that H_d is described by $\alpha(v' - v_2)^T = 0$. But then $f_P(v_2) \cap H_d = f_P(v') \cap H_d = f_3$, which is a contradiction to $f_2 \subset f_3$.

We come to the second point. W.l.o.g. let the intersection $H_d \cap \mathcal{P}_d$ be describable by an equation of the form

$$\alpha_1 = x_2 \cdot \alpha_2 + x_3 \cdot \alpha_3 + \dots + x_{d-1} \cdot \alpha_{d-1} + b =: f(\alpha)$$

with $x_i, b \in \mathbb{R}$. We map each point $v := (v_1, v_2, \dots, v_d) \in P$ to

$$\eta(v) := (v_2 + v_1 \cdot \tilde{v}_{2,1} + v_d \cdot \tilde{v}_{2,d}, v_3 + v_1 \cdot \tilde{v}_{3,1} + v_d \cdot \tilde{v}_{3,d}, \dots, v_d + v_1 \cdot \tilde{v}_{d,1} + v_d \cdot \tilde{v}_{d,d})$$

with $\tilde{v}_{i,1} = x_i + b$, $\tilde{v}_{i,d} = -(x_i + b)$ and $x_d = 0$. Note that η reduces the number of dimensions by one. Let $\eta(P)$ be the set of mapped points of P . By definition, $\mathcal{S}_{d-1}(\eta(P))$ has at most $\varphi_{d-1}^{d-2}(n, l)$ $(d - 2)$ -dimensional facets. Thus, there are at most $\varphi_{d-1}^{d-2}(n, l)$ extreme shortest paths in $\eta(P)$.

Let the map $\beta : H_d \cap \mathcal{P}_d \rightarrow \mathcal{P}_{d-1}$ be defined as follows.

$$\beta(\alpha) = (\alpha_2, \alpha_3, \dots, \alpha_d + \alpha_1)$$

For any $v \in P$ and $\alpha \in H_d \cap \mathcal{P}_d$ we have

$$\begin{aligned} \alpha \cdot v^T &= \left(f(\alpha), \alpha_2, \dots, 1 - f(\alpha) - \sum_{1 < i < d} \alpha_i \right) \cdot (v_1, \dots, v_d)^T \\ &= f(\alpha)(v_1 - v_d) + v_d + \sum_{1 < i < d} \alpha_i(v_i - v_d) \\ &= b(v_1 - v_d) + (\alpha_1 + \alpha_d)v_d + \sum_{1 < i < d} \alpha_i x_i(v_1 - v_d) + \sum_{1 < i < d} \alpha_i v_i \\ &= (\alpha_1 + \alpha_d)(v_d + v_1 \tilde{v}_{d,1} + v_d \tilde{v}_{d,d}) + \sum_{1 < i < d} \alpha_i(v_i + v_1 \tilde{v}_{i,1} + v_d \tilde{v}_{i,d}) \\ &= \beta(\alpha) \eta(v)^T. \end{aligned}$$

3.3 Decomposing Path Sets

In this section we first look at path sets Π_{sut} that can be written as the pairwise concatenation of two path sets Π_{su} and Π_{ut} that end/start at a common node u . We will see that in such a scenario the PSS $\mathcal{S}(P(\Pi_{sut}, c))$ is the overlay of $\mathcal{S}(P(\Pi_{su}, c))$ and $\mathcal{S}(P(\Pi_{ut}, c))$ for any cost function c (see Figure 4 for an example). Thus, we can upper bound $\varphi_d^{d-2}(\Pi_{sut})$ with the help of Lemma 12.

► **Lemma 13.** *Let Π_{su} and Π_{ut} be two path sets such that each path in Π_{su} ends at node $u \in V$ and each path in Π_{ut} starts at node u . Furthermore, let $\Pi_{sut} := \{\pi_1 \pi_2 \mid \pi_1 \in \Pi_{su}, \pi_2 \in \Pi_{ut}\}$ be the pairwise concatenation of Π_{su} and Π_{ut} . Then it holds*

$$\varphi_d^{d-2}(\Pi_{sut}) \leq \varphi_{d-1}^{d-2}(\Pi_{ut}) \cdot \varphi_d^{d-2}(\Pi_{su}) + \varphi_{d-1}^{d-2}(\Pi_{su}) \cdot \varphi_d^{d-2}(\Pi_{ut}).$$

Proof. We fix an arbitrary cost function $c \in \mathcal{C}_d$ and prove the inequality for c . Let $P_{sut} := P(\Pi_{sut}, c)$, $P_{su} := P(\Pi_{su}, c)$ and $P_{ut} := P(\Pi_{ut}, c)$. Given any $\alpha \in \mathcal{P}_d$, let v_{su} and v_{ut} be the α -shortest paths in P_{su} and P_{ut} . Then with $v_{sut} := v_{su} + v_{ut}$ it holds $v_{sut} \in P_{sut}$ and v_{sut} is the α -shortest path in P_{sut} . This is true because all paths in Π_{sut} go via node u . Thus, $\mathcal{S}(P_{sut})$ is the overlay of $\mathcal{S}(P_{su})$ and $\mathcal{S}(P_{ut})$.

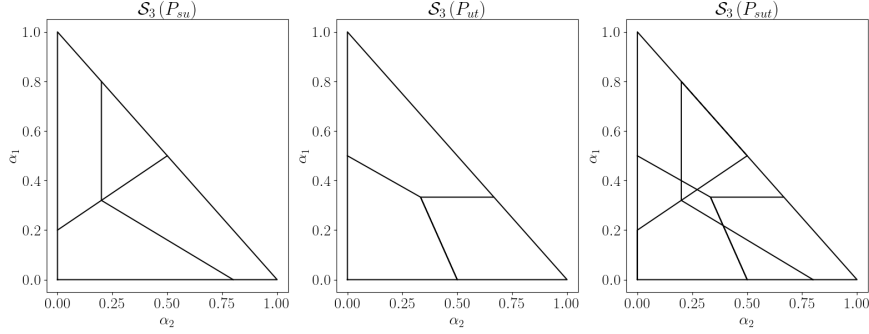
Every facet $f_{P_{su}}^{d-2}$ is part of a hyperplane H_d . If we let H_d intersect the preference subdivision $\mathcal{S}_d(P_{ut})$ we know from Lemma 12 that this intersection contains no more than $\varphi_{d-1}^{d-2}(\Pi_{ut})$ $(d-2)$ -dimensional facets. Thus, in the overlay of $\mathcal{S}(P_{su})$ and $\mathcal{S}(P_{ut})$ the facet $f_{P_{su}}^{d-2}$ can be split into at most $\varphi_{d-1}^{d-2}(\Pi_{ut})$ $(d-2)$ -dimensional facets as well. An analogous statement holds for the facets $f_{P_{ut}}^{d-2}$, which finishes the proof. ◀

The following lemma addresses the problem of upper bounding $\varphi_d^{d-2}(\Pi')$ if the path set Π' is the union of multiple path sets.

► **Lemma 14.** *Given k path sets $\Pi_1, \Pi_2, \dots, \Pi_k$ and let $\Pi' := \bigcup_{1 \leq i \leq k} \Pi_i$, then it holds for every $d > 1$*

$$\varphi_d^{d-2}(\Pi') \leq 2 \cdot k \cdot \varphi' \cdot \sum_{1 \leq i \leq k} \varphi_d^{d-2}(\Pi_i),$$

with $\varphi' := \max_{1 \leq i \leq k} \varphi_{d-1}^{d-2}(\Pi_i)$.



■ **Figure 4** This figure illustrates the meaning of Lemma 13. With the two path sets Π_{su} and Π_{ut} the PSS $\mathcal{S}(P_{sut})$, as defined in Lemma 13, is the overlay of $\mathcal{S}(P_{su})$ and $\mathcal{S}(P_{ut})$. An edge $f_{P_{su}}^1$ can cause multiple edges in the PSS $\mathcal{S}_3(P_{sut})$ by intersecting the PSS $\mathcal{S}_3(P_{ut})$. Lemma 13 says that $f_{P_{su}}^1$ can be split into at most $\varphi_2^2(\Pi_{ut})$ many edges in $\mathcal{S}_3(P_{sut})$.

Proof. We fix an arbitrary cost function $c \in \mathcal{C}_d$ and define $P' := P(\Pi', c)$ and $P_i := P(\Pi_i, c)$ for all $1 \leq i \leq k$.

Now, let us fix a facet $f_{P'}^{d-2} \in \mathcal{S}(P')$. The facet is part of a hyperplane H that is described by $\alpha(v_1 - v_2)^T = 0$ with $v_1, v_2 \in \mathcal{M}(P')$ and $v_1 \neq v_2$.

We first assume that v_1 and v_2 belong to the same set P_i . As $P_i \subseteq P'$, we have $v_1, v_2 \in \mathcal{M}(P_i)$ and $f_{P'}(v_1) \cap f_{P'}(v_2) \subseteq f_{P_i}(v_1) \cap f_{P_i}(v_2)$. Therefore, $\dim(f_{P_i}(v_1) \cap f_{P_i}(v_2)) = d-2$ and there is a facet $f_{P_i}^{d-2} = f_{P_i}(v_1) \cap f_{P_i}(v_2)$. Thus, we have at most $\varphi_d^{d-2}(\Pi_i)$ such pairs v_1, v_2 in $\mathcal{M}(P_i)$.

We now assume that v_1 and v_2 belong to different sets P_i and P_j . From $P_i \cup P_j \subseteq P'$ it follows that $f_{P'}(v_1) \cap f_{P'}(v_2) \subseteq f_{P_i}(v_1) \cap f_{P_j}(v_2)$. Thus, $\dim(f_{P_i}(v_1) \cap f_{P_j}(v_2)) \geq d-2$.

Therefore, there must be a facet $f_{P_i}^{d-2}$ or $f_{P_j}^{d-2}$ (or both) that intersects with $f_{P_i}(v_1) \cap f_{P_j}(v_2)$.

In summary, for each facet $f_{P'}^{d-2}$ we either find a corresponding facet $f_{P_i}^{d-2}$ with $f_{P'}^{d-2} \subseteq f_{P_i}^{d-2}$ (case one) or an intersection $f_{P_i}^{d-2} \cap f_{P_j}^{d-2}$ (or $f_{P_i}^{d-1} \cap f_{P_j}^{d-2}$) with $\dim(f_{P_i}^{d-2} \cap f_{P_j}^{d-2}) = d-2$ (case two). Clearly, we can upper bound the first case with $\sum_{1 \leq i \leq k} \varphi_d^{d-2}(\Pi_i)$.

The second case we handle with the help of Lemma 12. The intersection of the facet $f_{P_i}^{d-2}$ with any PSS $\mathcal{S}_d(P_j)$ can contain at most $\varphi_{d-1}^{d-2}(\Pi_j) \leq \varphi'(d-2)$ -dimensional facets (Lemma 12). Moreover, there are at most two extreme shortest paths in P_i that are adjacent to $f_{P_i}^{d-2}$. Thus, in total the facet $f_{P_i}^{d-2}$ can contribute to at most $2 \cdot (k-1) \cdot \varphi'$ intersections of case two. Therefore, the number of facets $f_{P'}^{d-2}$ of case two can be upper bounded by $2 \cdot (k-1) \cdot \varphi' \cdot \sum_{1 \leq i \leq k} \varphi_d^{d-2}(\Pi_i)$.

With $\sum_{1 \leq i \leq k} \varphi_d^{d-2}(\Pi_i) + 2 \cdot (k-1) \cdot \varphi' \cdot \sum_{1 \leq i \leq k} \varphi_d^{d-2}(\Pi_i) \leq 2 \cdot k \cdot \varphi' \cdot \sum_{1 \leq i \leq k} \varphi_d^{d-2}(\Pi_i)$ the statement follows. ◀

3.4 Proving the Upper Bound via Recursion

In the following lemma we describe a recursive upper bound for $\varphi_d^{d-2}(n, l)$ in the dimension d , which we then use to prove Theorem 10.

► **Lemma 15.** $\varphi_d^{d-2}(n, l) \leq d \cdot l^2 \cdot n^{2 \log l} \cdot \varphi_{d-1}^{d-2}(n, l)^{2 \log l}$ for $d > 1$.

Proof. We prove Lemma 15 for an arbitrary complete path set $\Pi_{st}(l)$ and cost function $c \in \mathcal{C}_d$ and define $P_{st} := P(\Pi_{st}(l), c)$. We first introduce path sets $\Pi_{sut} := \{\pi \in \Pi_{st}(l) \mid u \in V \text{ divides } \pi \text{ into subpaths of at most } \lceil \frac{l}{2} \rceil \text{ edges}\}$. Clearly, each path $\pi \in \Pi_{st}(l)$ is contained in at least one path set Π_{sut} . Thus, we can apply Lemma 14 and get

$$\varphi_d^{d-2}(\Pi_{st}(l)) \leq 2 \cdot n \cdot \varphi_{d-1}^{d-2}(n, l) \cdot \sum_{u \in V} \varphi_d^{d-2}(\Pi_{sut}), \quad (7)$$

as one can easily show that $\varphi_{d-1}^{d-2}(n, l) \geq \max_{u \in V} \varphi_{d-1}^{d-2}(\Pi_{sut})$.

The path sets Π_{sut} can be written as the pairwise concatenation of the path sets $\Pi_{su}(\frac{l}{2})$ and $\Pi_{ut}(\frac{l}{2})$. Thus, they satisfy the requirements to apply Lemma 13 such that for each node u we get

$$\begin{aligned} \varphi_d^{d-2}(\Pi_{sut}) &\leq \varphi_{d-1}^{d-2}\left(\Pi_{ut}\left(\frac{l}{2}\right)\right) \cdot \varphi_d^{d-2}\left(\Pi_{su}\left(\frac{l}{2}\right)\right) + \varphi_{d-1}^{d-2}\left(\Pi_{su}\left(\frac{l}{2}\right)\right) \cdot \varphi_d^{d-2}\left(\Pi_{ut}\left(\frac{l}{2}\right)\right) \\ &\leq 2 \cdot \varphi_{d-1}^{d-2}\left(n, \frac{l}{2}\right) \cdot \varphi_d^{d-2}\left(n, \frac{l}{2}\right). \end{aligned} \quad (8)$$

If we combine (7) and (8), we get

$$\begin{aligned} \varphi_d^{d-2}(\Pi_{st}(l)) &\leq 4 \cdot n^2 \cdot \varphi_{d-1}^{d-2}(n, l) \cdot \varphi_{d-1}^{d-2}\left(n, \frac{l}{2}\right) \cdot \varphi_d^{d-2}\left(n, \frac{l}{2}\right) \\ &\leq 4 \cdot n^2 \cdot \varphi_{d-1}^{d-2}(n, l)^2 \cdot \varphi_d^{d-2}\left(n, \frac{l}{2}\right). \end{aligned} \quad (9)$$

Using this recursion in l we obtain with $\varphi_d^{d-2}(n, 1) = d$

$$\begin{aligned} \varphi_d^{d-2}(\Pi_{st}(l)) &\leq d \cdot 4^{\log l} \cdot n^{2 \log l} \cdot \varphi_{d-1}^{d-2}(n, l)^{2 \log l} \\ &= d \cdot l^2 \cdot n^{2 \log l} \cdot \varphi_{d-1}^{d-2}(n, l)^{2 \log l}. \end{aligned} \quad (10)$$

Proof of Theorem 10. We need to show that $\varphi_d^{d-1}(n, n) \in n^{O(\log^{d-1} n)}$ for a fixed but arbitrary number of cost types $d \geq 1$. From Lemma 15 we know that $\varphi_d^{d-2}(n, n) \leq d \cdot n^{2+2 \log n} \cdot \varphi_{d-1}^{d-2}(n, n)^{2 \log n}$. With (5) we get $\varphi_d^{d-1}(n, n) \leq 2 \cdot n^{2+2 \log n} \cdot \varphi_{d-1}^{d-2}(n, n)^{2 \log n}$. With $\varphi_1^0(n, n) = 1$ this leads to $\varphi_d^{d-1}(n, n) \in n^{O((2 \log n)^{d-1})}$, which is, for a fixed d , equal to $\varphi_d^{d-1}(n, n) \in n^{O(\log^{d-1} n)}$. ◀

4 Conclusions

In this paper we showed that the number of extreme shortest paths in a graph G is upper bounded by $n^{O(\log^{d-1} n)}$, where n is the number of nodes and d is the fixed but arbitrary number of edge costs in G . This is a generalization of previous results in the context of parametric shortest paths for two and three dimensions.

An open question is whether one can also generalize the matching two-dimensional lower bounds shown in [5].

References

- 1 Florian Barth, Stefan Funke, and Claudius Proissl. Preference-based trajectory clustering: An application of geometric hitting sets. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
- 2 Florian Barth, Stefan Funke, and Sabine Storandt. Alternative multicriteria routes. In *2019 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 66–80, 2019. doi:10.1137/1.9781611975499.6.

- 3 HP Benson and E Sun. Outcome space partition of the weight set in multiobjective linear programming. *Journal of Optimization Theory and Applications*, 105(1):17–36, 2000.
- 4 Jon Louis Bentley, Hsiang-Tsung Kung, Mario Schkolnick, and Clark D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, October 1978. doi:10.1145/322092.322095.
- 5 Patricia J Carstensen. Complexity of some parametric integer and network programming problems. *Mathematical Programming*, 26(1):64–75, 1983.
- 6 Patricia June Carstensen. *The complexity of some problems in parametric linear and combinatorial programming*. PhD thesis, University of Michigan, 1983.
- 7 Daniel Delling and Dorothea Wagner. Pareto paths with SHARC. In *Proc. 8th International Symposium on Experimental Algorithms (SEA)*, volume 5526 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2009.
- 8 Stefan Funke and Sabine Storandt. Personalized route planning in road networks. In *Proc. 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, Bellevue, WA, USA*, pages 45:1–45:10. ACM, 2015.
- 9 Kshitij Gajjar and Jaikumar Radhakrishnan. Parametric shortest paths in planar graphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 876–895. IEEE, 2019.
- 10 Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route planning with flexible objective functions. In *Workshop on Algorithms and Experimentation, ALENEX '10*, pages 124–137, USA, 2010. Society for Industrial and Applied Mathematics.
- 11 Daniel Mier Gusfield. *Sensitivity analysis for combinatorial optimization*. PhD thesis, University of California, Berkeley, 1980.
- 12 Gabriel Y Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.
- 13 Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. Route skyline queries: A multi-preference path planning approach. In *26th International Conference on Data Engineering (ICDE)*, pages 261–272. IEEE Computer Society, 2010.
- 14 Kurt Mehlhorn and Mark Ziegemann. Resource constrained shortest paths. In *European Symposium on Algorithms*, pages 326–337. Springer, 2000.
- 15 Ketan Mulmuley and Pradyut Shah. A lower bound for the shortest path problem. In *Proceedings 15th Annual IEEE Conference on Computational Complexity*, pages 14–21. IEEE, 2000.

Galactic Token Sliding

Valentin Bartier ✉

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Nicolas Bousquet ✉ 

Univ Lyon, CNRS, UCBL, INSA Lyon, LIRIS, UMR5205, France

Amer E. Mouawad ✉ 

American University of Beirut, Lebanon

University of Bremen, Germany

Abstract

Given a graph G and two independent sets I_s and I_t of size k , the INDEPENDENT SET RECONFIGURATION problem asks whether there exists a sequence of independent sets (each of size k) $I_s = I_0, I_1, I_2, \dots, I_\ell = I_t$ such that each independent set is obtained from the previous one using a so-called reconfiguration step. Viewing each independent set as a collection of k tokens placed on the vertices of a graph G , the two most studied reconfiguration steps are token jumping and token sliding. In the TOKEN JUMPING variant of the problem, a single step allows a token to jump from one vertex to any other vertex in the graph. In the TOKEN SLIDING variant, a token is only allowed to slide from a vertex to one of its neighbors. Like the INDEPENDENT SET problem, both of the aforementioned problems are known to be $W[1]$ -hard on general graphs (for parameter k). A very fruitful line of research [5, 14, 27, 25] has showed that the INDEPENDENT SET problem becomes fixed-parameter tractable when restricted to sparse graph classes, such as planar, bounded treewidth, nowhere-dense, and all the way to biclique-free graphs. Over a series of papers, the same was shown to hold for the TOKEN JUMPING problem [17, 22, 26, 8]. As for the TOKEN SLIDING problem, which is mentioned in most of these papers, almost nothing is known beyond the fact that the problem is polynomial-time solvable on trees [11] and interval graphs [6]. We remedy this situation by introducing a new model for the reconfiguration of independent sets, which we call galactic reconfiguration. Using this new model, we show that (standard) TOKEN SLIDING is fixed-parameter tractable on graphs of bounded degree, planar graphs, and chordal graphs of bounded clique number. We believe that the galactic reconfiguration model is of independent interest and could potentially help in resolving the remaining open questions concerning the (parameterized) complexity of TOKEN SLIDING.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → W hierarchy

Keywords and phrases reconfiguration, independent set, galactic reconfiguration, sparse graphs, token sliding, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.15

Related Version *Full Version:* <https://arxiv.org/abs/2204.05549>

Funding This work is supported by PHC Cedre project 2022 “PLR”.

Valentin Bartier: Supported by ANR project GrR (ANR-18-CE40-0032).

Nicolas Bousquet: Supported by ANR project GrR (ANR-18-CE40-0032).

Amer E. Mouawad: Research supported by the Alexander von Humboldt Foundation and partially supported by URB project “A theory of change through the lens of reconfiguration”.

1 Introduction

Many algorithmic questions can be posed as follows: given the description of a system state and the description of a state we would “prefer” the system to be in, is it possible to transform the system from its current state into the more desired one without “breaking” the system in the process? And if yes, how many steps are needed? Such problems



© Valentin Bartier, Nicolas Bousquet, and Amer E. Mouawad;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 15; pp. 15:1–15:14

Leibniz International Proceedings in Informatics



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

naturally arise in the fields of mathematical puzzles, operational research, computational geometry [23], bioinformatics, and quantum computing [13] for instance. These questions received a substantial amount of attention under the so-called *combinatorial reconfiguration framework* in the last few years [9, 28, 30]. We refer the reader to the surveys by van den Heuvel [28] and Nishimura [24] for more background on combinatorial reconfiguration.

In this work, we focus on the reconfiguration of independent sets. Given a simple undirected graph G , a set of vertices $S \subseteq V(G)$ is an *independent set* if the vertices of S are all pairwise non-adjacent. Finding an independent set of maximum cardinality, i.e., the INDEPENDENT SET problem, is a fundamental problem in algorithmic graph theory and is known to be not only NP-hard, but also W[1]-hard and not approximable within $\mathcal{O}(n^{1-\epsilon})$, for any $\epsilon > 0$, unless $P = NP$ [31].

We view an independent set as a collection of tokens placed on the vertices of a graph such that no two tokens are adjacent. This gives rise to two natural adjacency relations between independent sets (or token configurations), also called *reconfiguration steps*. These two reconfiguration steps, in turn, give rise to two combinatorial reconfiguration problems. In the TOKEN JUMPING (TJ) problem, introduced by Kamiński et al. [20], a single reconfiguration step consists of first removing a token on some vertex u and then immediately adding it back on any other vertex v , as long as no two tokens become adjacent. The token is said to *jump* from vertex u to vertex v . In the TOKEN SLIDING (TS) problem, introduced by Hearn and Demaine [15], two independent sets are adjacent if one can be obtained from the other by a token jump from vertex u to vertex v with the additional requirement of uv being an edge of the graph. The token is then said to *slide* from vertex u to vertex v along the edge uv . Note that, in both the TJ and TS problems, the size of independent sets is fixed. Generally speaking, in the TOKEN JUMPING and TOKEN SLIDING problems, we are given a graph G and two independent sets I_s and I_t of G . The goal is to determine whether there exists a sequence of reconfiguration steps – a *reconfiguration sequence* – that transforms I_s into I_t (where the reconfiguration step depends on the problem).

Both problems have been extensively studied, albeit under different names [6, 7, 11, 12, 16, 19, 20, 22]. It is known that both problems are PSPACE-complete, even on restricted graph classes such as graphs of bounded bandwidth (and hence pathwidth) [29] and planar graphs [15]. On the positive side, it is easy to prove that TOKEN JUMPING can be decided in polynomial time on trees (and even on chordal graphs) since we simply have to iteratively jump tokens to leaves (resp. vertices that only appear in the bag of a leaf in the clique tree) to transform an independent set into another. Unfortunately, for TOKEN SLIDING, the problem becomes more complicated because of what we call the *bottleneck effect*. Indeed, there might be a lot of empty leaves in the tree but there might be a bottleneck in the graph that prevents us from reaching these desirable vertices. For instance, consider a star plus a long subdivided path attached to the center of the star. One cannot move any token from the leaves of the star to the path if there are at least two tokens on the leaves (in other words, two tokens adjacent to a cut-vertex prevent us from using the cut vertex). Even if we can overcome this issue for instance on trees [11] and on interval graphs [6], the TOKEN SLIDING problem remains much “harder” than the TOKEN JUMPING problem. In split graphs for instance (which are chordal), TOKEN SLIDING is PSPACE-complete [4]. Lokshtanov and Mouawad [21] showed that, in bipartite graphs, TOKEN JUMPING is NP-complete while TOKEN SLIDING remains PSPACE-complete.

In this paper we focus on the parameterized complexity of the TOKEN SLIDING problem. While the complexity of TOKEN JUMPING parameterized by the size of the independent set is quite well understood, the comprehension of the complexity of TOKEN SLIDING remains

evasive. A problem Π is FPT (fixed-parameterized tractable) parameterized by k if one can solve it in time $f(k) \cdot \text{poly}(n)$, for some computable function f . In other words, the combinatorial explosion can be restricted to a parameter k . In the rest of the paper, our parameter k will be the size of the independent set (i.e. number of tokens). Both TOKEN JUMPING and TOKEN SLIDING are known to be W[1]-hard¹ parameterized by k on general graphs [22]. On the positive side, Lokshantov et al. [22] showed that TOKEN JUMPING is FPT on bounded degree graphs. This result has been extended in a series of papers to planar graphs, nowhere-dense graphs, and finally strongly $K_{\ell,\ell}$ -free graphs [18, 8], a graph being strongly $K_{\ell,\ell}$ -free if it does not contain any $K_{\ell,\ell}$ as a subgraph. For TOKEN SLIDING, it was proven in [2] that the problem is W[1]-hard on bipartite graphs and C_4 -free graphs (a similar result holds for TOKEN JUMPING but based on weaker assumptions for the bipartite case [1]).

Almost no positive result is known for TOKEN SLIDING even for incredibly simple cases like graphs of bounded degree. Our main contributions are to develop two general tools for the design of parameterized algorithms for TOKEN SLIDING, namely galactic reconfiguration and types. Galactic reconfiguration is a general simple tool that allows us to reduce instances. Using it, we will derive that TOKEN SLIDING is FPT on bounded degree graphs. Our second tool, called types, will in particular permit to show that when the deletion of a small subset of vertices leaves too many components, then one of them can be removed. Combining both tools with additional rules, we prove that TOKEN SLIDING is FPT on planar graphs and on chordal graphs of bounded clique number. We complement these results by proving that TOKEN SLIDING is W[1]-hard on split graphs.

Our first result is the following:

► **Theorem 1.1.** *TOKEN SLIDING is FPT on bounded degree graphs parameterized by k .*

Much more than the result itself, we believe that our main contribution here is the general framework we developed for its proof, namely galactic reconfiguration. Before explaining exactly what it is, let us explain the intuition behind it. As we already said, even if there are vertices which are far apart from the vertices of an independent set, we are not sure we can reach them because of the bottleneck effect. Our intuition was that it should be possible to reduce a part of large diameter of the graph that does not contain any tokens (just as we can find irrelevant vertices when we have large grid minors). The idea is that since the diameter is large, we should be able to hide tokens far apart from each other in this structure, avoiding the “bottleneck issue”. And thus the structure should be reducible. However, proving that a structure can be reduced in reconfiguration is usually very technical. To overcome this problem, we introduce a new kind of vertices called *black holes* which can swallow as many tokens of the independent set as we like. A *galactic graph* is a graph that might contain black holes. A *galactic independent set* is a set of vertices on which tokens lie, such that the set of non black hole vertices holding tokens is an independent set and such that each black hole might contain any number of tokens.

Our main result is to prove that if there exists a long shortest path that is at distance at least two from the initial and target independent sets, then we can replace it by a black hole (whose neighborhood is the union of the neighborhoods of the path vertices). This rule, together with other simple rules on galactic graphs, allows us to reduce the size of bounded-degree graphs until they reach a size of at most $f(k)$ in polynomial time. Since a kernel ensures the existence of an FPT algorithm, Theorem 1.1 holds.

¹ Informally, it means that they are very unlikely to admit an FPT algorithm.

In the rest of the paper, we combine galactic graphs with other techniques to prove that TOKEN SLIDING is FPT on several other graph classes. We first prove the following:

► **Theorem 1.2.** *TOKEN SLIDING is FPT on planar graphs parameterized by k .*

To prove Theorem 1.2, we cannot simply use our previous long path reduction since, in a planar graph, there might be a universal vertex which prevents the existence of a long shortest path. Note that the complexity of TOKEN SLIDING is open on outerplanar graphs and it was not known to be FPT prior to our work.

Our strategy consists in reducing to planar graphs of bounded degree and then applying Theorem 1.1. To do so, we provide another general tool to reduce graphs for TOKEN SLIDING. Namely, we show that if there is a set X of vertices such that $G - X$ contains too many connected components (in terms of k and $|X|$) then at least one of them can be safely removed. The idea of the proof consists in defining the *type* of a connected component of $G - X$. From a very high level perspective, the type² of a path in a component of $G - X$ is the sequence of its neighborhoods in X . The type of a component C is the union of the types of the paths starting on a vertex of C . We then show that if too many components of $G - X$ have the same type then one of them can be removed.

However, this component reduction is not enough since, in the case of a vertex x universal to an outerplanar graph, the deletion of x does not leave many connected components. We prove that, we can also reduce a planar graph if (i) there are too many vertex-disjoint (x, y) -paths for some pair x, y of vertices or, (ii) if a vertex has too many neighbors on an induced path. Since one can prove that in an arbitrarily large planar graph with no long shortest path (i) or (ii) holds, it will imply Theorem 1.2.

Note that our proof techniques can be easily adapted to prove that the problem is FPT for any graph of bounded genus. We think that the notion of types may be crucial to derive FPT algorithms on larger classes of graphs such as bounded treewidth graphs.

We finally provide another application of our method by proving that the following holds:

► **Theorem 1.3.** *TOKEN SLIDING is FPT on chordal graphs of bounded clique number.*

The proof of Theorem 1.3 consists in proving that, since there is a long path in the clique tree, we can either find a long shortest path (and we can reduce the graph using galactic rules) or find a vertex x in a large fraction of the bags of this path. In the second case, we show that we can again reduce the graph. We complement this result by proving that it cannot be extended to split graphs, contrarily to TOKEN JUMPING.

► **Theorem 1.4.** *TOKEN SLIDING is $W[1]$ -hard on split graphs.*

We show hardness via a reduction from the MULTICOLORED INDEPENDENT SET problem, known to be $W[1]$ -hard [10]. The crux of the reduction relies on the fact that we have a clique of unbounded size and hence we can use different subsets of the clique to encode vertex selection gadgets and non-edge selection gadgets.

The first natural generalization of our result on chordal graphs of bounded clique size would be the following:

► **Question 1.1.** *Is TOKEN SLIDING FPT on bounded treewidth graphs? Or simpler, on bounded pathwidth graphs?*

² The exact definition is actually more complicated.

We did not succeed in answering Question 1.1 but we think that the method we used for Theorem 1.3 is a good starting point (with a much more involved analysis). Recall that the problem is PSPACE-complete on graphs of constant bandwidth for a large enough constant that is not explicit in the proof [29]. Note that our galactic reconfiguration rules directly ensure that TOKEN SLIDING is FPT on bounded bandwidth graphs and the multi-component reduction ensures that the problem is FPT for graphs of bounded treedepth. But even for bounded pathwidth, the situation is unclear. There are good indications to think that solving the bounded pathwidth case is the hardest step to obtain an FPT algorithm for bounded treewidth graphs. On the positive side, we simply know that the problem is polynomial time solvable on graphs of treewidth one (namely forests) [11] and the problem is open for outerplanar graphs, which are graphs of treewidth 2:

► **Question 1.2.** *Is TOKEN SLIDING polynomial-time solvable on outerplanar graphs? Triangulated outerplanar graphs?*

Organization of the paper. In Section 2, we formally introduce galactic graphs and provide our main reduction rules concerning such graphs, including the long shortest path reduction lemma. In Section 3, we introduce the notion of types and journeys and prove that if there are too many connected components in $G - X$ then at least one of them can be removed. In Section 4, we briefly describe our results for TOKEN SLIDING on planar graphs and chordal graphs of bounded clique number. Our hardness reduction for split graphs and all omitted proofs can be found in the full version of the paper [3].

2 Galactic graphs and galactic token sliding

We say that a graph $G = (V, E)$ is a *galactic graph* when $V(G)$ is partitioned into two sets $A(G)$ and $B(G)$ where the set $A(G) \subseteq V(G)$ is the set of vertices that we call *planets* and the set $B(G) \subseteq V(G)$ is the set of vertices that we call *black holes*. For a given graph G' , we write $G' \prec G$ whenever $|A(G')| < |A(G)|$ or, in case of equality, $|B(G')| < |B(G)|$. In the standard TOKEN SLIDING problem, tokens are restricted to sliding along edges of a graph as long as the resulting sets remain independent. This implies that no vertex can hold more than one token and no two tokens can ever become adjacent. In a galactic graph, the rules of the game are slightly modified. When a token reaches a black hole (a special kind of vertex), the token is *absorbed* by the black hole. This implies that a black hole can hold more than one token, in fact it can hold all k tokens. Moreover, we allow tokens to be adjacent as long as one of the two vertices is a black hole (since black holes are assumed to make tokens “disappear”). On the other hand, a black hole can also “project” any of the tokens it previously absorbed onto any vertex in its neighborhood (be it a planet or a black hole). Of course, all such moves require that we remain an independent set in the galactic sense. We say that a set I is a *galactic independent set* of a galactic graph G whenever $G[I \cap A]$ is edgless. To fully specify a galactic independent set I of size k containing more than one token on black holes, we use a weight function $\omega_I : V(G) \rightarrow \{0, \dots, k\}$. Hence, $\omega_I(v) \leq 1$ whenever $v \in A(G)$, $\omega_I(v) \in \{0, \dots, k\}$ whenever $v \in B(G)$, and $\sum_{v \in V(G)} \omega_I(v) = k$.

We are now ready to define the GALACTIC TOKEN SLIDING problem. We are given a galactic graph G , an integer k , and two galactic independent sets I_s and I_t such that $|I_s| = |I_t| = k \geq 2$ (when $k = 1$ the problem is trivial). The goal is to determine whether there exists a sequence of token slides that will transform I_s into I_t such that each intermediate set remains a galactic independent set. As for the classical TOKEN SLIDING problem, given a galactic graph G we can define a reconfiguration graph which we call the *galactic*

reconfiguration graph of G . It is the graph whose vertex set is the set of all galactic independent sets of G , two vertices being adjacent if their corresponding galactic independent sets differ by exactly one token slide. We always assume the input graph G to be a connected graph, since we can deal with each component independently otherwise. Furthermore, components without tokens can be safely deleted. Given an instance (G, k, I_s, I_t) of GALACTIC TOKEN SLIDING, we say that (G, k, I_s, I_t) can be *reduced* if we can find an instance (G', k', I'_s, I'_t) which is positive (yes-instance) if and only if (G, k, I_s, I_t) is positive and $G' \prec G$.

Let G be a galactic graph. A *planetary component* is a maximal connected component of $G[A]$. A *planetary path* P , or *A-path*, composed only of vertices of A , is called *A-geodesic* if, for every x, y in P , $\text{dist}_{G[A]}(x, y) = \text{dist}_P(x, y)$. We use the term *A-distance* to denote the length of a shortest path between vertices $u, v \in A$ such that all vertices of the path are also in A . Let us state a few reduction rules that allow us to safely reduce an instance (G, k, I_s, I_t) of GALACTIC TOKEN SLIDING to an instance (G', k', I'_s, I'_t) .

- Reduction rule R1 (adjacent black holes rule): If two black holes u and v are adjacent, we contract them into a single black hole w . If there are tokens on u or v , the merged black hole receives the union of all such tokens. In other words, $\omega_{I'_s}(w) = \omega_{I_s}(u) + \omega_{I_s}(v)$ and $\omega_{I'_t}(w) = \omega_{I_t}(u) + \omega_{I_t}(v)$. Loops and multi-edges are ignored.
- Reduction rule R2 (dominated black hole rule): If there exists two black holes u and v such that $N(u) \subseteq N(v)$, $\omega_{I_s}(u) = 0$, and $\omega_{I_t}(u) = 0$, we delete u .
- Reduction rule R3 (absorption rule): If there exists u, v such that u is a black hole, $v \in N(u) \cap A$ (v is a neighboring planet that could be in $I_s \cup I_t$) and $|((I_s \cup I_t) \cap A) \cap N[v]| \leq 1$, then we contract the edge uv . We say that v is *absorbed* by u . If $v \in I_s \cup I_t$ then we update the weights of u accordingly.
- Reduction rule R4 (twin planets rule): Let $u, v \in A(G)$ be two planet vertices that are *twins* (true or false twins). That is, either $uv \notin E(G)$ and $N(u) = N(v)$ or $uv \in E(G)$ and $N[u] = N[v]$. If $u \notin I_s \cup I_t$ then delete u . If both u and v are in I_s (resp. I_t) and at least one of them is not in I_t (resp. I_s) then return a trivial no-instance. If both u and v are in I_s as well as I_t then delete $N[u] \cup N[v]$, decrease k by two, and set $I'_s = I_s \setminus \{u, v\}$ and $I'_t = I_t \setminus \{u, v\}$.
- Reduction rule R5 (path reduction rule): Let G be a galactic graph and P be a A -geodesic path of length at least $5k$ such that $(A \cap N[P]) \cap (I_s \cup I_t) = \emptyset$. Then, P can be contracted into a black hole (we ignore loops and multi-edges). That is, we contract all edges in P until one vertex remains.

► **Lemma 2.1.** *Reduction rule R5, the path reduction rule, is safe.*

Sketch of the proof. Let P be an A -geodesic path of length $5k$ in G such that no vertex of $A \cap N[P]$ are in the initial or target independent sets, I_s and I_t . Let G' be the graph obtained after contracting P into a single black hole b . Let I'_s and I'_t be the galactic independent sets corresponding to I_s and I_t . If there is a transformation from I_s to I_t in G , then one can show that there is a transformation from I'_s to I'_t in G' by simply absorbing tokens that become adjacent to the black hole and then projecting them appropriately when needed.

Now we consider a transformation from I'_s to I'_t in G' and show how to adapt it in G . We first prove (in the full version of the paper [3]) that we can always assume the existence of a sequence in G' where the number of tokens in $N(b) \cap A$ is at most one throughout the sequence, for any black hole b . Now, assuming such a sequence, we can simulate the sequence in G with the hard case being when multiple tokens slide into b . Note, however, that P is of length $5k$ and is A -geodesic. Hence, every vertex $a \in A$ has at most three neighbors in P and any independent set of size at most k in A has at most $3k$ neighbors in P . This leaves

$2k$ vertices on P which we can use to hold as many as k tokens that need to slide into b (in G'). In other words, whenever more than one token slides into b in G' , we simulate this by sliding the tokens in P onto the $2k$ vertices of P that are free. ◀

As immediate consequences, the following properties hold in an instance where reduction rules R1 to R5 cannot be applied.

► **Corollary 2.2.** *Every planetary component must contain at least one token and therefore G can have at most k planetary components, when $k \geq 2$.*

► **Corollary 2.3.** *Let (G, k, I_s, I_t) be an instance of GALACTIC TOKEN SLIDING where reduction rules R1, R3, and R5 (adjacent black holes rule, absorption rule, and the path reduction rule) have been exhaustively applied. Then, the graph G has diameter at most $O(k^2)$. Moreover, any planetary component has diameter at most $O(k^2)$.*

We now show how the galactic reconfiguration framework combined with the previous reduction rules immediately implies that TOKEN SLIDING is fixed-parameter tractable for parameter $k + \Delta(G)$, where $\Delta(G)$ denotes the maximum degree of G . Theorem 2.4 immediately implies positive results for graphs of bounded bandwidth/bucketwidth.

► **Theorem 2.4.** *TOKEN SLIDING is fixed-parameter tractable when parameterized by $k + \Delta(G)$. Moreover, the problem admits a bikernel³ with $k\Delta(G)^{O(k^2)} + (2k + 2k\Delta(G))\Delta(G)$ vertices.*

Proof. Let (G, k, I_s, I_t) be an instance of TOKEN SLIDING. We first transform it to an instance of GALACTIC TOKEN SLIDING where all vertices are planetary vertices. We then apply all of the reduction rules R1 to R5 exhaustively. By a slight abuse of notation we let (G, k, I_s, I_t) denote the irreducible instance of GALACTIC TOKEN SLIDING.

The total number of planetary components in G is at most k by Corollary 2.2 and the diameter of each such component is at most $O(k^2)$ by Corollary 2.3. Hence the total number of planet vertices is at most $k\Delta(G)^{O(k^2)}$.

To bound the total number of black holes, it suffices to note that no black hole can have a neighbor in $B \cup (A \setminus N[I_s \cup I_t])$. In other words, no black hole can be adjacent to another black hole (since the adjacent black holes reduction rule would apply) and no black hole can be adjacent to a planet without neighboring tokens (otherwise the absorption reduction rule would apply). Hence, combined with the fact that each black hole must have degree at least one, the total number of black holes is at most $(2k + 2k\Delta(G))\Delta(G)$. ◀

3 The multi-component reduction rule (R6)

General idea. The goal of this section is to show how we can reduce a graph when we have a small vertex separator with many components attached to it. We let X be a subset of vertices and H be an induced subgraph of $G - X$ (for simplicity we assume G is a non-galactic graph in this section). Let I_s and I_t be two independent sets which are disjoint from H and consider a reconfiguration sequence from I_s to I_t in G . Let v be a vertex of H and assume that there is a token t that is *projected* on v at some point of the reconfiguration sequence, meaning that the token t is moved from a vertex of X to v . This token may stay a few steps on v , move to some other vertex w of H , and so on until it eventually goes back to X . Let this sequence of vertices (allowing duplicate consecutive vertices) be denoted by $v_1 = v, v_2, \dots, v_r$. We call this sequence the *journey* of v (formal definitions are given in the next subsection).

³ A kernel where the resulting instance is not an instance of the same problem.

Assume now that the number of connected components attached to X is arbitrarily large. Our goal is to show that one of those components can be safely deleted, that is, without compromising the existence of a reconfiguration sequence if one exists. Suppose that we decide to delete the component H . The transformation from I_s to I_t does not exist anymore since, in the reconfiguration sequence, the token t was projected on $v \in V(H)$. But we can ask the following question: Is it possible to simulate the journey of v in another connected component of $G - X$? In fact, if we are able to find a vertex w in a connected component $H' \neq H$ of $G - X$ and a sequence $w_1 = w, \dots, w_r$ of vertices such that $w_i w_{i+1}$ is an edge for every i and such that $N(w_i) \cap X = N(v_i) \cap X$, then we could project the token t on w instead of v and perform this journey instead of the original journey⁴ of t . One possible issue is that the number r of (distinct) vertices in the journey can be arbitrarily large, and thus the existence of w and H' is not guaranteed *a priori*. This raises more questions: What is important in the sequence $v = v_1, \dots, v_r$? Why do we go from v_1 to v_r ? Why many steps in the journey if r is large? The answers are not necessarily unique. We distinguish two cases.

First, suppose that in the reconfiguration sequence, the token t was projected from X to v , performed the journey without having to “wait” at any step (so no duplicate consecutive vertices in the journey), and then was moved to a vertex $x' \in X$. Then, the journey only needs to “avoid” the neighbors of the vertices in X that contain a token. Let us denote by s_1 the step where the token t is projected on v and by s_2 the last step of the journey (that is, the step where t is one move/slide away from X). Let Y be the vertices of X that contain a token between the steps s_1 and s_2 . The journey of t can then be summarized as follows: a vertex whose neighborhood in X is equal to $N(v) \cap X$, a walk whose vertices all belong to H and are only adjacent to subsets of $X \setminus Y$, and then a vertex whose neighborhood in X is equal to $N(v_r) \cap X$. In particular, if we can find, in another connected component of $G - X$, a vertex w for which such a journey (with respect to the neighborhood in X) also exists, then we can project t on w instead of v . Clearly, the obtained reconfiguration sequence would also be feasible (assuming again no other tokens in the component of w).

However, we might not be able to go “directly” from $v_1 = v$ to v_r . Indeed, at some point in the sequence, there might be a vertex v_{i_1} which is adjacent to a token in X . This token will eventually move (since the initial journey with t in H is valid), which will then allow the token t to go further on the journey. But then again, either we can reach the final vertex v_r or the token t will have to wait on another vertex v_{i_2} for some token on X to move, and so on (until the end of the journey). We say that there are *conflicts*⁵ during the journey⁵.

So we can now “compress” the path as a path from v_1 to v_{i_1} , then from v_{i_1} to v_{i_2} (together with the neighborhood in X of these paths), as we explained above. However, we cannot yet claim that we have reduced the instance sufficiently since the number of conflicts is not known to be bounded (by a function of k and/or the size of X). The main result of this section consists in proving that, if we consider a transformation from I_s to I_t that minimizes the number of moves “related” to X , then (almost) all the journeys have a “controllable” amount of (so-called important) conflicts. Actually, we prove that, in most of the connected components H of $G - X$, we can assume that we have a “controllable” number of important conflicts for every journey on H in a transformation that minimizes the number of token modifications involving X . The idea consists in proving that, if there are too many important conflicts during a journey of a token t , we could mimic the journey of t on another component to reduce the number of token slides involving X . Finally, we will only have to prove that if all the vertices have a controllable number of conflicts (and there are too many components), then we can safely delete a connected component of $G - X$.

⁴ We assume for simplicity in this outline that the component of w does not contain tokens.

⁵ Actually, there might exist another type of conflict we do not explain in this outline for simplicity.

Journeys and conflicts. We denote a reconfiguration sequence from I_s to I_t by $\mathcal{R} = \langle I_0, I_1, \dots, I_{\ell-1}, I_\ell \rangle$. Let $X \subseteq V(G)$ and H be a component in $G - X$ such that $I_s \cap V(H) = I_t \cap V(H) = \emptyset$. All along this section, we are assuming that tokens have labels just so we can keep track of them. For every token t , let $v_i(t)$, $0 \leq i \leq \ell$, denote the vertex on which token t is at position i in the reconfiguration sequence \mathcal{R} .

Whenever a token enters H and leaves it, we say that the token makes a journey in H . Let I_i denote the first independent set in \mathcal{R} where $v_i(t) \in V(H)$ and let I_j , $i \leq j$, denote the first independent set after I_i where $v_{j+1}(t) \notin V(H)$. Then the *journey J of t in H* is the sequence $(v_i(t), \dots, v_j(t))$. The journey is a sequence of vertices (with multiplicity) from H such that consecutive vertices are either the same or connected by an edge. We associate each journey J with a walk W in H . The *walk W of t in H* is the journey of t where duplicate consecutive vertices have been removed.

We say that a token is *waiting at step i* if $v_i(t) = v_{i-1}(t)$; otherwise the token is *active*. Given a journey J and its associated walk W , we say that $w \in W$ is a *waiting vertex* if there is a step where the vertex w is a waiting vertex in the journey. Otherwise w is an *active vertex* (with respect to the reconfiguration sequence). So we can now decompose the walk W into waiting vertices and transition walks. That is, assuming the walk starts at y and ends at z , we can write $W = yP_0w_0P_1w_1 \dots w_\ell P_\ell z$, where each w_i is a waiting vertex and each P_i is a *transition walk* (consisting of the walk of active vertices between two consecutive waiting vertices). Note that the transition walks could be empty.

We are interested in why a token t might be waiting at some vertex w . In fact, we will only care about waiting vertices that we will call important waiting vertices. Let w_1, \dots, w_ℓ be the waiting vertices of the journey and, for every $i \leq \ell$, let us denote by $[s_i, s'_i]$ the time interval of the reconfiguration sequence where the token t is staying on the vertex w_i . Note that $s_i < s'_i$ and when t is active the other tokens are not moving; thus the position of any token different from t is the same all along the interval $[s'_i + 1, s_{i+1}]$ for every $i \leq \ell - 1$.

Let $i < j \leq \ell$ and let w_i be a waiting vertex. We say that w_j is the *important waiting vertex after w_i* if $j > i$ and j is the largest integer such that no vertex along the walk of token t between w_i (included) and w_j (included) is adjacent to a token $t' \neq t$ or contains a token $t' \neq t$ between steps s'_i and s_j (note that the important waiting vertex after w_i might be the last vertex of the sequence). Since token t is active from $s'_i + 1$ to s_{i+1} and is moving from w_i to w_{i+1} during that interval, the important waiting vertex after w_i is well-defined and is at least w_{i+1} . Let $Q_{i,j}$ denote the walk in H that the token t follows to go from w_i to w_j (both w_i and w_j are included in $Q_{i,j}$). In other words, $Q_{i,j} = w_i P_{i+1} w_{i+1} \dots P_j w_j$. Now, note that since w_j is the important waiting vertex after w_i (i.e. we cannot replace w_j by w_{j+1}), then we claim that the following holds:

- ▷ **Claim 3.1.** If w_j is not the last vertex of the walk W , either
- (i) there is a token on or adjacent to a vertex of $P_{j+1} w_{j+1}$ (the transition walk after w_j) at some step in $[s'_i, s'_j]$ or,
 - (ii) there is a token on or adjacent to a vertex of $Q_{i,j} - w_j$ in the interval $[s_j, s_{j+1}]$.

We now define the notion of conflicts. Since we cannot replace w_j by w_{j+1} , it means that, by definition, there is at least one step s_q in $[s'_i, s_{j+1}]$ where a token $t_q \neq t$ is adjacent to (or on a vertex) v_q of $Q_{i,j+1}$. We call such a step a *conflict*. We say that (s_q, v_q, t_q) is the *conflict triplet* associated to the conflict (we will mostly refer to a triplet as a conflict).

The conflicts of type (i) are called *right conflicts* and the conflicts of type (ii) are called *left conflicts*. It might be possible that w_j is the important waiting vertex because we have (several) left and right conflicts. We say that w_j is a *left important vertex* if there is at least one left conflict and a *right important vertex* otherwise.

If w_j is a left important vertex, we let the *important conflict* (s_*, v_*, t_*) denote the first conflict associated with $Q_{i,j}$ between steps s_j and s'_j , i.e., there exists no s such that $s_j \leq s < s_* \leq s'_j$ such that there is a conflict at step s with a token $t' \neq t$ which is either on $Q_{i,j}$ or incident to $Q_{i,j}$. Note that v_* cannot be a vertex of $Q_{i,j}$ since that would imply at least one more conflict before s_* , hence $v_* \in N(V(Q_{i,j}))$. If w_j is a right important vertex, we let (s_*, v_*, t_*) denote the *important conflict* associated with $P_{j+1}w_{j+1}$ between steps s'_i and s'_j as the last conflict associated to $P_{j+1}w_{j+1}$, i.e., there exists no s such that $s'_i \leq s_* < s \leq s'_j$ and there is a conflict (s, v_s, t_s) such that v_s in $P_{j+1}w_{j+1}$ or incident to $P_{j+1}w_{j+1}$. Note that v_* cannot be a vertex of $P_{j+1}w_{j+1}$ since that would imply at least one more conflict after s_* , hence $v_* \in N(V(P_{j+1}w_{j+1}))$. We use $\mathcal{C}(Q_{i,j+1})[s'_i, s'_j]$ to denote all conflict triplets (left and right conflicts) associated with $Q_{i,j+1}$ between steps s'_i and s'_j .

To conclude this section, let us remark that the conflicts might be due to vertices of H or vertices of X . In other words, for a conflict triple $(s, v, t) \in \mathcal{C}(Q_{i,j+1})[s'_i, s'_j]$, v is an H -conflict or an X -conflict depending on whether v is in H or in X . In what follows we will only be interested in X -conflicts. The *X-important waiting vertex after w_i* is w_j where $j > i$ is the smallest integer such that $\mathcal{C}(Q_{i,j+1})[s'_i, s'_j]$ contains at least one triplet (s, v, t') where $t' \neq t$, $v \in X$, and $s'_i \leq s \leq s'_j$. Now given a journey we can define the sequence of X -important waiting vertices as the sequence w'_1, \dots, w'_r starting with vertex w_1 and such that w'_{j+1} is the X -important waiting vertex after w_j . What will be important in the rest of the section is the length r of this sequence. If this sequence is short (bounded by $f(k)$), then we can check if we can simulate a similar journey in other components efficiently. If the sequence is long, we will see that it implies that we can find a “better” transformation.

Since we will mostly be interested in how a journey interacts with X , we introduce the notion of the X -walk associated with journey J . The X -walk is written as $W^X = yP_0w_0P_1w_1 \dots w_\ell P_\ell z$, where each w is an X -important waiting vertex and each P is the walk that the token takes (this walk could have non-important waiting vertices) before reaching the next X -important waiting vertex. We call each P in an X -walk an *X-transition walk*.

Types and signatures. Let X be a subset of vertices and H be a component of $G - X$. An ℓ -type is defined as a sequence $IY_1W_1Y_2W_2 \dots Y_\ell W_\ell Y_{\ell+1}F$ such that for every i , W_i is a (possibly empty) subset of X and Y_i is a (possibly empty) subset of X or a special value \perp (the meaning of \perp will become clear later on). We call I the *initial value* and F the *final value* and they are both non-empty subsets of vertices of X . The 0-type is defined as IY_0F and we allow I to be equal to F . We will often represent an ℓ -type by $(I(Y_iW_i)_{i \leq \ell} Y_{\ell+1}F)$. Note that if X is bounded, then the number of ℓ -types is bounded. More precisely, we have:

► **Remark 3.2.** The number of ℓ -types is at most $(2^{|X|} + 1)^{2(\ell+2)}$.

The neighborhood of a set of vertices $S \subseteq V(H)$ in X is called the *X-trace* of S . A journey J is *compatible* with an ℓ -type $IY_1W_1Y_2W_2 \dots Y_\ell W_\ell Y_{\ell+1}F$ if it is possible to partition the X -walk W of J into $W^X = yP_0w_0P_1w_1 \dots P_\ell w_\ell P_{\ell+1}z$ such that:

- the X -trace of each vertex w_i is W_i ,
- for every walk P_i which is not empty, the X -trace of P_i is Y_i , i.e., $\cup_{x \in P_i} N(x) \cap X = Y_i$ (note that we can have $Y_i = \emptyset$),
- for every empty walk P_i , we have $Y_i = \perp$, and
- the X -trace of y is I and the X -trace of z is F .

The ℓ -signature of a vertex $v \in V(H)$ (with respect to X) is the set of all ℓ' -types with $\ell' \leq \ell$ that can be *simulated* by v in H . That is, for every ℓ -type, there exists a walk W starting at v such that $W = vP_0w_0P_1w_1 \dots P_\ell w_\ell P_{\ell+1}z$ is compatible with the ℓ -type if and only if the ℓ -type is in the signature. Two vertices are *ℓ -equivalent* if their ℓ -signatures are the same.

► **Lemma 3.3.** *One can compute in $O^*((2^{|X|} + 1)^{2(\ell+2)})$ the ℓ -signature of a vertex v in H .*

X-reduced sequences and equivalent journeys. Let J be a journey with exactly r X -important waiting vertices (in its X -walk). Let w_i and P_i be respectively the i -th X -important waiting vertex and the i -th X -transition walk. Let P_{r+1} be the final X -transition walk. Let us denote by W_i the neighborhoods of w_i in X , and by Y_i the neighborhood of P_i in X . Let I and F be the neighborhoods of the initial and final vertices of the walk associated with J , respectively. The *type* T of the journey J is $I(Y_i W_i)_{i \leq r} Y_{r+1} F$.

► **Definition 3.4.** *Two journeys are X -equivalent whenever the following holds:*

- *They have the same number of X -important waiting vertices;*
- *The initial and final vertex of the X -walk have the same X -trace;*
- *For every i , the X -trace of the i th X -important waiting vertex is the same in both journeys;*
- *For every i , the X -trace of the i th X -transition walk is the same in both journeys.*

Let I, J be two independent sets and X be a subset of vertices of G . A slide of a token *is related to* X if the token is moving from or to a vertex in X (possibly from some other vertex in X). We call such a move an X -move.

► **Definition 3.5.** *A transformation \mathcal{R} from I to J is X -reduced if the number of X -moves is minimized and, among the transformations that minimize the number of X -moves, \mathcal{R} minimizes the total number of moves.*

The multi-component reduction. Let H be a connected component of $G - X$. The ℓ -signature of H is the union of the ℓ -signatures of the vertices in H . Let \mathcal{H} be a subset of connected components of $G - X$. We say that $H \in \mathcal{H}$ is ℓ -dangerous for \mathcal{H} if there is a ℓ -type in the ℓ -signature of H that appears in at most ℓ connected components of \mathcal{H} . Otherwise we say that H is ℓ -safe. If there are no ℓ -dangerous components, we say that \mathcal{H} is ℓ -safe. One can easily prove the following using iterated extractions:

► **Lemma 3.6.** *Let $\ell = 5|X|k$. If there are more than $\ell(2^{|X|} + 1)^{2(\ell+2)} + 2k + 1$ components in $G - X$, then there exists a collection of at least $2k + 1$ components that are ℓ -safe which can be found in $f(k, |X|) \cdot n^{O(1)}$ -time, for some computable function f .*

We can now prove the main result of this section:

► **Lemma 3.7.** *Let I_s, I_t be two independent sets and X be a subset of $V(G)$. Let \mathcal{R} be an X -reduced transformation from I_s to I_t . Assume that there exists a subset \mathcal{H} of at least $2k + 1$ connected components of $G - X$ that is $(5|X|k)$ -safe. Then, for every $C \in \mathcal{H}$, any journey on the component C has at most $5|X|k - 1$ X -important waiting vertices in its X -walk.*

Sketch of the proof. Assume for a contradiction that there exists a safe component $C \in \mathcal{H}$ and a journey J of some token t in C that has at least $5|X|k$ X -important waiting vertices. Amongst all such journeys, select the one that reaches first its $(5|X|k)$ -th X -important waiting vertex. Let us denote by $I(Y_i W_i)_{i \leq 5k} Y_{5k+1}$ the type of the journey J that we truncate after Y_{5k+1} . And, let us denote by $v(P_i w_i)_{i \leq 5k} P_{5k+1}$ the partition of the walk into X -important waiting vertices and X -transition walks (we assume the walk starts at vertex $v \in V(C)$).

For each X -important waiting vertex w_i , let (q_i, x_i, t_i) be the important conflict associated to it. Since there are at most k labels of tokens and $|X|$ vertices in X , there exists a vertex $x \in X$ and a token with label t' such that there exists at least 5 waiting vertices such that the important conflict is of the form (q, x, t') for some q . In other words, there exists

15:12 Galactic Token Sliding

P_{i_1}, \dots, P_{i_5} such that for each P_i we have a triplet (q, x, t') (recall that q denotes the step in the reconfiguration sequence). Let us denote by s_1, \dots, s_5 the steps of those conflicts whose token label is t' and whose vertex in X is x and such that these conflicts are important conflict triples in five different X -transition walks.

The rest of the proof consists in proving that, rather than making the initial transformation, we can project t' on an appropriately chosen component C' of \mathcal{H} distinct from C and mimic the journey of t between steps s_1 and s_5 . In other words, t' can simply follow a journey on C' of the same type as that of t between s_1 and s_5 and we can then safely project it back on x at step s_5 without creating any X -conflicts. This implies that we can strictly reduce the number of X -conflicts, a contradiction to the assumption that the transformation is X -reduced. We also manage H -conflicts by using the minimality of the journey. ◀

► **Lemma 3.8.** *Let I_s, I_t be two independent sets and X be a subset of $V(G)$. If $G - X$ contains at least $4k + 2$ $(5|X|k)$ -safe components, then we can delete one of those components, say C , such that there is a transformation from I_s to I_t in G if and only if there is a transformation in $G - V(C)$.*

Sketch of the proof. The proof consists in proving that, since all the journeys have at most $5|X|k - 1$ X -important waiting vertices by Lemma 3.7, one can replace a journey in a safe component by a journey in another component either by not creating any conflicts or by creating some conflicts on shorter “time periods” which ensures the procedure converges. ◀

► **Corollary 3.9.** *Given a cutset X , we can assume that $G - X$ has at most $5|X|k(2^{|X|} + 1)^{2(5|X|k+2)} + 4k + 2 = 2^{O(|X|^2k)}$ connected components. Moreover, when the number of components is larger we can find a component to delete in $f(k, |X|) \cdot n^{O(1)}$ -time.*

4 Applications

Due to space restriction we only very briefly explain the main steps of the proofs.

Planar graphs. First note that, using galactic reconfiguration, we can reduce the diameter of the graph to $O(k^2)$ by Corollary 2.3. The core of the proof then consists in reducing high degree vertices. If the degree is bounded by a function of k , then the conclusion will follow from Theorem 2.4. To reduce the degree, we prove that, for every pair x, y of large degree vertices, 1) $V(G) \setminus \{x, y\}$ does not contain too many components by Corollary 3.9 (Rule R6), 2) the graph can be reduced if there are not too many internally vertex-disjoint paths from x to y and, 3) the graph can be reduced if it contains a few other slightly more general structures. All these results together permit to ensure that G does not contain any large degree vertices after applying all the rules, which completes the proof.

Chordal graphs of bounded clique number. A *chordal graph* is a graph with no induced cycle of length at least four. Equivalently, it also has a clique-tree (see full version [3] for formal definitions). If a node v of the clique-tree has large degree then $V(G) \setminus B_v$ (where B_v denotes the bag of v) has many connected components, and one of them can be removed by Corollary 3.9. So we can assume that the degree of the clique-tree is bounded. Thus, if the graph is large, the clique-tree should contain a long path P . If no vertex belongs to a large fraction of the bags of P , then the diameter is large and we can reduce the graph by Corollary 2.3. So a vertex belongs to a large fraction of the bags of P . We can actually prove that, there exists a sub-path P' of P and $X \subseteq V(G)$ such that X belongs to all the bags of P' and no other vertex appears in a significant fraction of the bags of P' . Together with a few other properties, we prove that some vertices in the bags of P' can be deleted without modifying the existence of a reconfiguration sequence from I_s to I_t .

References

- 1 Akanksha Agrawal, Ravi Kiran Allumalla, and Varun Teja Dhanekula. Refuting FPT algorithms for some parameterized problems under gap-eth. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPICs*, pages 2:1–2:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.2.
- 2 Valentin Bartier, Nicolas Bousquet, Clément Dallard, Kyle Lomer, and Amer E. Mouawad. On girth and the parameterized complexity of token sliding and token jumping. *Algorithmica*, 83(9):2914–2951, 2021. doi:10.1007/s00453-021-00848-1.
- 3 Valentin Bartier, Nicolas Bousquet, and Amer E. Mouawad. Galactic token sliding. *CoRR*, abs/2204.05549, 2022. doi:10.48550/arXiv.2204.05549.
- 4 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 13:1–13:17, 2019. doi:10.4230/LIPICs.STACS.2019.13.
- 5 Hans L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In G. Goos, J. Hartmanis, D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, Timo Lepistö, and Arto Salomaa, editors, *Automata, Languages and Programming*, volume 317, pages 105–118. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988. Series Title: Lecture Notes in Computer Science. doi:10.1007/3-540-19488-6_110.
- 6 Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. 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 127–139. Springer, 2017. doi:10.1007/978-3-319-68705-6_10.
- 7 Paul S. Bonsma, Marcin Kaminski, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, pages 86–97, 2014.
- 8 Nicolas Bousquet, Arnaud Mary, and Aline Parreau. Token Jumping in Minor-Closed Classes. In Ralf Klasing and Marc Zeitoun, editors, *Fundamentals of Computation Theory*, volume 10472, pages 136–149. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-662-55751-8_12.
- 9 Richard C. Brewster, Sean McGuinness, Benjamin Moore, and Jonathan A. Noel. A dichotomy theorem for circular colouring reconfiguration. *Theor. Comput. Sci.*, 639:1–13, 2016.
- 10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 11 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theoretical Computer Science*, 600:132–142, October 2015. doi:10.1016/j.tcs.2015.07.037.
- 12 Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pages 237–247, 2015.
- 13 Sevag Gharibian and Jamie Sikora. Ground state connectivity of local hamiltonians. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 617–628, 2015. doi:10.1007/978-3-662-47672-7_50.
- 14 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding First-Order Properties of Nowhere Dense Graphs. *Journal of the ACM*, 64(3):1–32, June 2017. doi:10.1145/3051095.

- 15 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.
- 16 Takehiro Ito, Marcin Kamiński, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. On the parameterized complexity for token jumping on graphs. In *Theory and Applications of Models of Computation - 11th Annual Conference, TAMC 2014, Chennai, India, April 11-13, 2014. Proceedings*, pages 341–351, 2014.
- 17 Takehiro Ito, Marcin Kamiński, and Hirotaka Ono. Fixed-Parameter Tractability of Token Jumping on Planar Graphs. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation*, volume 8889, pages 208–219. Springer International Publishing, Cham, 2014. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-319-13075-0_17.
- 18 Takehiro Ito, Marcin Kamiński, and Hirotaka Ono. Fixed-parameter tractability of token jumping on planar graphs. In *Algorithms and Computation*, Lecture Notes in Computer Science, pages 208–219. Springer International Publishing, 2014.
- 19 Takehiro Ito, Hiroyuki Nooka, and Xiao Zhou. Reconfiguration of vertex covers in a graph. *IEICE Transactions*, 99-D(3):598–606, 2016.
- 20 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.
- 21 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms*, 15(1):7:1–7:19, 2019. doi:10.1145/3280825.
- 22 Daniel Lokshtanov, Amer E. Mouawad, Fahad Panolan, M.S. Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. *Journal of Computer and System Sciences*, 95:122–131, August 2018. doi:10.1016/j.jcss.2018.02.004.
- 23 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point set is NP-complete. *Comput. Geom.*, 49:17–23, 2015. doi:10.1016/j.comgeo.2014.11.001.
- 24 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.
- 25 Michał Pilipczuk and Sebastian Siebertz. Kernelization and approximation of distance- r independent sets on nowhere dense graphs. *European Journal of Combinatorics*, 94:103309, May 2021. doi:10.1016/j.ejc.2021.103309.
- 26 Sebastian Siebertz. Reconfiguration on Nowhere Dense Graph Classes. *The Electronic Journal of Combinatorics*, 25(3):P3.24, August 2018. doi:10.37236/7458.
- 27 J.A. Telle and Y. Villanger. FPT algorithms for domination in sparse graphs and beyond. *Theoretical Computer Science*, 770:62–68, May 2019. doi:10.1016/j.tcs.2018.10.030.
- 28 Jan van den Heuvel. The complexity of change. *Surveys in Combinatorics 2013*, 409:127–160, 2013.
- 29 Marcin Wrochna. Reconfiguration in bounded bandwidth and treedepth. *CoRR*, 2014. arXiv:1405.0847.
- 30 Marcin Wrochna. Homomorphism reconfiguration via homotopy. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 730–742, 2015.
- 31 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.

When Are Cache-Oblivious Algorithms Cache Adaptive? A Case Study of Matrix Multiplication and Sorting

Arghya Bhattacharya ✉ 🏠
Stony Brook University, NY, USA

Helen Xu ✉
Lawrence Berkeley National Laboratory,
CA, USA

Rezaul A. Chowdhury ✉ 🏠
Stony Brook University, NY, USA

Rishab Nithyanand ✉
The University of Iowa, Iowa City, IA, USA

Abiyaz Chowdhury ✉
Stony Brook University, NY, USA

Rathish Das ✉
University of Waterloo, Canada

Rob Johnson ✉
VMware Research, Palo Alto, CA, USA

Michael A. Bender ✉ 🏠
Stony Brook University, NY, USA

Abstract

Cache-adaptive algorithms are a class of algorithms that achieve optimal utilization of dynamically changing memory. These memory fluctuations are the norm in today’s multi-threaded shared-memory machines and time-sharing caches.

Bender et al. [8] proved that many cache-oblivious algorithms are optimally cache-adaptive, but that some cache-oblivious algorithms can be relatively far from optimally cache-adaptive on worst-case memory fluctuations. This worst-case gap between cache obliviousness and cache adaptivity depends on a highly-structured, adversarial memory profile. Existing cache-adaptive analysis does not predict the relative performance of cache-oblivious and cache-adaptive algorithms on non-adversarial profiles. Does the worst-case gap appear in practice, or is it an artifact of an unrealistically powerful adversary?

This paper sheds light on the question of whether cache-oblivious algorithms can effectively adapt to realistically fluctuating memory sizes; the paper focuses on matrix multiplication and sorting. The two matrix-multiplication algorithms in this paper are canonical examples of “ (a, b, c) -regular” cache-oblivious algorithms, which underlie much of the existing theory on cache-adaptivity. Both algorithms have the same asymptotic I/O performance when the memory size remains fixed, but one is optimally cache-adaptive, and the other is not. In our experiments, we generate both adversarial and non-adversarial memory workloads. The performance gap between the algorithms for matrix multiplication grows with problem size (up to $3.8\times$) on the adversarial profiles, but the gap does not grow with problem size (stays at $2\times$) on non-adversarial profiles. The sorting algorithms in this paper are not “ (a, b, c) -regular,” but they have been well-studied in the classical external-memory model when the memory size does not fluctuate. The relative performance of a non-oblivious (cache-aware) sorting algorithm degrades with the problem size: it incurs up to $6\times$ the number of disk I/Os compared to an oblivious adaptive algorithm on both adversarial and non-adversarial profiles.

To summarize, in all our experiments, the cache-oblivious matrix-multiplication and sorting algorithms that we tested empirically adapt well to memory fluctuations. We conjecture that cache-obliviousness will empirically help achieve adaptivity for other problems with similar structures.

2012 ACM Subject Classification Theory of computation → Shared memory algorithms

Keywords and phrases Cache-adaptive algorithms, cache-oblivious algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.16

Supplementary Material *Software*: https://github.com/ArghyaB118/cache_adaptivity

Funding This research in part was funded by NSF grant CCF-1617618, CCF-1439084, CCF-1725543, the Canada Research Chairs Program, NSERC Discovery Grants, NSF grant CNS-1553510.

This research is funded in part by the Advanced Scientific Computing Research (ASCR) program



© Arghya Bhattacharya, Abiyaz Chowdhury, Helen Xu, Rathish Das, Rezaul A. Chowdhury, Rob Johnson, Rishab Nithyanand, and Michael A. Bender; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 16; pp. 16:1–16:17

Leibniz International Proceedings in Informatics



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

within the Office of Science of the DOE under contract number DE-AC02-05CH11231, the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

This research was sponsored in part by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator 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.

1 Introduction

Applications running on multi-threaded and multi-core systems often experience fluctuations in the amount of physical memory available. For example, when concurrently running programs share RAM, the amount of RAM allocated to any particular program can change as other programs start or finish. Interestingly, even when identical copies of the same program run concurrently, these copies may acquire an unequal fraction of the RAM [20], resulting in further unpredictability.

These memory fluctuations can cause an application's performance to suffer [24, 26]. If the available memory becomes scarce, an application may **thrash**, which means that the application pages excessively, crippling performance [18, 19, 28]. If memory becomes abundant, the program may not take advantage of the larger available RAM and thus perform extraneous I/Os.

In this paper, we empirically show that some optimal cache-oblivious algorithms [21, 22, 31] – specifically matrix multiplication and sorting – adapt gracefully to most memory fluctuations, except for some adversarially constructed worst-case instances. Optimal **cache-oblivious** algorithms are asymptotically optimal for all fixed memory sizes without the knowledge of the memory size. Past theoretical work shows that optimal cache-oblivious algorithms can perform poorly on adversarially constructed memory fluctuations [7, 8].

Experimental approaches to adaptivity. Past experimental work [29, 30, 34, 35] focused on developing improved algorithms that adapt to a variety of memory profiles, where a **memory profile** is a curve representing the memory size as a function of time. For example, Pang et al. [29] introduced a memory-adaptive merge sort that dynamically adjusts the size of each subproblem at each level of the recursion as memory fluctuates. A similar technique is used by Pang et al. [30] to modify the GRACE hash join algorithm [23] so that it can adapt to changes in memory size. Zhang and Larson [34, 35] introduced techniques that balance the memory usage among many sorting programs running concurrently on a single machine sharing its memory.

Researchers have also explored environment-level modifications to adapt to dynamic memory profiles. For example, Brown et al. [12] introduced memory-management techniques for a DBMS that accommodate multiple concurrently running database workloads, where each workload may have its own individual memory requirements. Mills et al. [24–26] proposed a user interface that enables a user to request sufficient memory for their application.

Theoretical approaches to adaptivity. Barve and Vitter [2, 3] designed algorithms that have provable performance guarantees even when the memory size changes. They gave memory-adaptive algorithms for sorting, matrix multiplication, FFT, LU decomposition,

and permutation. To prove optimality, they extended the traditional **external-memory model** [1] – also called the **disk-access machine (DAM)** – to allow for changes in the memory size over time.

More recently, Bender et al. [8] showed that many cache-oblivious algorithms could adapt well to memory fluctuations. A **cache-oblivious algorithm** is a platform-independent algorithm that is not parameterized by properties of the memory hierarchy such as RAM or cache-line size [21, 22, 31]. An optimal cache-oblivious algorithm is universal in the sense that the algorithm is optimal for any value of the cache parameters, provided that these values do not change over time. (Past theoretical work on cache-oblivious algorithms use the terms memory, cache, and RAM interchangeably) [4, 5]. On the other hand, a cache-adaptive algorithm remains cache-optimal even when memory size changes over time. One possible way to deal with memory fluctuations is to start with an algorithm already known to be cache-oblivious and hope it also happens to be cache adaptive.

Cache-obliviousness and cache-adaptivity. Bender et al. [8] showed that cache-oblivious algorithms are often (but not always) cache adaptive. For example, they showed that Lazy Funnel Sort (LFS) [10, 21], a cache-oblivious sorting algorithm, is optimally cache-adaptive. Follow-up work [7] provided an analytical framework for determining whether cache-oblivious algorithms having a certain recursive form (“ (a, b, c) -regular”) are also cache adaptive. Algorithms with (a, b, c) -regular recursive structure have a common form of divide-and-conquer cache-oblivious design [6, 7]; see Section 2 for more details. Depending on the settings of the parameters a , b , and c , these cache-oblivious algorithms are either asymptotically optimal or a logarithmic factor away from optimal. For example, cache-oblivious algorithms exist for matrix multiplication [21, 22], some of which are optimally cache-adaptive (MM-INPLACE) and some are a log factor away from optimally cache-adaptive (MM-SCAN) [7, 8]. MM-SCAN ($a = 8$, $b = 4$, and $c = 1$) performs an out-of-place matrix addition at the end of each recursive call and is suboptimal in the cache-adaptive model, while MM-INPLACE ($a = 8$, $b = 4$, and $c < 1$) performs the additions in-place and is optimally cache-adaptive; see Section 3.

Connections between experimental and theoretical approaches. Existing theoretical approaches to cache adaptivity provide worst-case guarantees but leave open the question of how algorithms perform under arbitrary memory workloads. Specifically, past work showed that cache-oblivious (a, b, c) -regular algorithms are always at most a log factor away from being optimally cache-adaptive. However, this log factor is based on a worst-case memory workload that seems brittle and unlikely to appear in practice. Furthermore, this worst-case analysis does not capture how cache-oblivious algorithms actually perform under practical memory workloads.

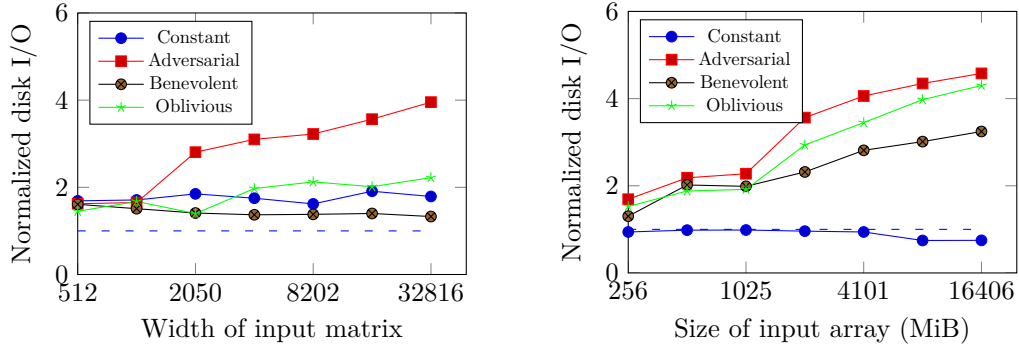
In contrast, empirical solutions [12, 23–26, 29, 30, 34, 35] are designed to improve performance under practical memory workloads. However, they lack worst-case analysis and therefore do not have worst-case performance guarantees.

Recent work [6] has gone beyond worst-case analysis. The authors apply smoothing techniques [32] on memory profiles and explore how well cache-oblivious algorithms adapt to smoothed memory profiles. They observe that the I/O-performance gap between cache-obliviousness and cache-adaptivity disappears given sufficient smoothing. Their results leave open the question of whether the performance gap between cache-oblivious and cache-adaptive algorithms persists across a larger space of memory profiles. If the performance gap turns out to be only a theoretical artifact rarely seen in practice, cache-oblivious algorithms

16:4 When Are Cache-Oblivious Algorithms Cache Adaptive?

could be an exciting way to solve a wide range of problems with no known cache-adaptive solutions. Examples of such problems include Gaussian elimination [15], triangle counting [9], min-weight cycle [33], negative triangle detection and counting [33], replacement paths problem [33], etc.

Do these worst-case memory workloads arise naturally when no adversary controls the available memory? How feasible is it to construct such worst-case memory profiles?



(a) Normalized I/O of MM-SCAN to MM-INPLACE.

(b) Normalized I/O of EMS to LFS.

■ **Figure 1** Normalized disk I/O of the non-adaptive algorithm to the cache-adaptive algorithm as a function of input size for matrix multiplication and sorting for four memory profiles: adversarial (worst-case), benevolent, constant, and oblivious. An increase in the y-axis means that the non-adaptive algorithm performs progressively worse relative to the adaptive algorithm.

Results

We empirically study how cubic-time matrix multiplication (MM) algorithms and external-memory sorting algorithms adapt to a range of different memory profiles. MM is a particularly interesting test case because not only is it the canonical example of an (a, b, c) -regular algorithm but also we can find a pair of MM algorithms, both with the same I/O-complexity in the cache-oblivious model, where one is optimally cache-adaptive and the other is a logarithmic factor away from optimal. For the case of MM, we evaluate MM-INPLACE and MM-SCAN and find that both can adapt to a wide range of memory profiles even though MM-SCAN is not provably cache-adaptive. In contrast, for the case of sorting, we evaluate LFS [10, 21] and External-memory Merge Sort (EMS) [1], a non-oblivious and non-adaptive algorithm, and find that EMS does not adapt well to a wide range of memory fluctuations. Table 1 illustrates some of the key properties of the tested algorithms.

Since the worst-case profile is tightly coupled with an algorithm’s structure, we designed a **memory profile generator** that can simulate an adversarial by looking into a program’s execution. It can generate the worst-case **adversarial memory profile** by increasing the available memory when the program cannot benefit from the extra memory and decreasing the memory when the program would benefit. It can also generate non-adversarial profiles that follow a program’s execution.

First, we empirically study the MM-SCAN and MM-INPLACE algorithms for matrix multiplication, and we find that even though MM-SCAN is not optimally cache-adaptive, it adapts well to a wide range of memory fluctuations except the most adversarially constructed profiles. We measure how well an algorithm adapts to the memory fluctuations by measuring

■ **Table 1** Properties of the algorithms for cubic-time matrix multiplication and external-memory sorting studied in this paper.

<i>Algorithm</i>	<i>Cache-oblivious</i>	<i>Cache-adaptive</i>
MM-SCAN [21, 22]	✓	✗
MM-INPLACE [21, 22]	✓	✓
EM Merge Sort [1]	✗	✗
Lazy Funnel Sort [10]	✓	✓

the disk I/Os it incurs during its execution. MM-SCAN and MM-INPLACE are both theoretically optimal when the memory does not fluctuate; empirically, Figure 1a shows that MM-INPLACE performs roughly $1.8\times$ better than MM-SCAN across all problem sizes under a **constant memory profile** (fixed memory size). We used the memory profile generator to create profiles coupled with algorithm execution. Under the adversarial memory workload, MM-SCAN performs $1.6 - 3.8\times$ more disk I/Os than MM-INPLACE. Critically, the performance gap grows with the input size. We also created a **benevolent memory profile** that is tightly synchronized with the algorithm’s execution but non-adversarial in nature. Under the benevolent memory profile, MM-SCAN and MM-INPLACE perform roughly within $1.5\times$ of each other for all problem sizes. Under **oblivious memory profiles**, i.e., profiles that are not tightly coupled with the algorithms’ execution, we found that MM-SCAN incurs about $1.8\times$ more I/Os than MM-INPLACE for all problem sizes. These results suggest that MM-SCAN possesses almost all the benefits of cache-adaptivity.

Next, we evaluate the cache-oblivious Lazy Funnel Sort (LFS) [10, 21] and the non-oblivious External-memory Merge Sort (EMS) [1] (both are I/O-optimal when the memory is fixed), and we find that even though LFS has a computational overhead over EMS stemming from its cache-obliviousness; it outperforms EMS on a wide range of fluctuating memory profiles. Figure 1b suggests that External-memory Merge Sort has better I/O-performance than Lazy Funnel Sort for all problem sizes under the constant memory profile (roughly incurring $0.9\times$ disk I/Os on average). On the other hand, LFS adapts when the memory fluctuates, whereas EMS fails to adapt. EMS performs progressively worse than LFS as the input size increases under all types of fluctuating memory workloads (adversarial, benevolent, and oblivious), incurring $1.3 - 7\times$ more disk I/Os. Despite the computational overhead of cache-obliviousness, LFS performs significantly better than EMS under a range of fluctuating memory profiles.

These results suggest that for the problems of MM and sorting, cache-oblivious but non-adaptive algorithms adapt well, while non-oblivious algorithms do not. The tested cache-oblivious algorithms adapt well because they are agnostic to the cache size, while the cache-aware algorithms are optimized for specific cache sizes. We conjecture that cache-oblivious algorithms have most of the empirical benefits of cache adaptivity, and our results are consistent with the conjecture.

Paper overview. The rest of this paper is organized as follows. Section 2 reviews preliminaries about the cache-adaptive model and analysis necessary to understand the experimental design in this paper. Section 3 explains the memory profiles that we tested on. Section 4 describes the experimental setup and the algorithms’ performance with fixed memory. Section 5 describes the memory-profile generator and the algorithms’ performance under adaptively constructed memory profiles. Section 6 explores the algorithms’ performance under obliviously constructed memory profiles. Section 7 provides concluding remarks and future directions.

2 Cache-adaptive analysis

This section reviews fundamentals of cache-adaptive and cache-oblivious analysis. It also explains how (a, b, c) -regular algorithms play a key role in the analysis.

Cache-adaptive model. The cache-adaptive model is an extension of the disk-access machine (DAM) model [1], where the size of memory available to an algorithm can change. In the DAM model, the machine has a two-level memory hierarchy comprising a cache/memory of size M and a disk of unbounded size. Data is transferred between disk and memory in blocks of size B , called I/Os. The cache-adaptive model extends the traditional DAM model by allowing the memory size to be a function of time. Each I/O takes one time step and computation is modeled as free and instantaneous. At time t , the memory available to a program is $M(t)$, which can change after each time step.

Cache-oblivious algorithms and (a, b, c) -regularity. An algorithm is cache-oblivious [21, 22, 31] if it is not parameterized by M and B . An optimal cache-oblivious algorithm is universal, in the sense that it runs optimally in the DAM model for all possible (fixed) values of M and B .

Cache-oblivious algorithms with a particular kind of divide-and-conquer structure, are said to be (a, b, c) -**regular**. An algorithm is (a, b, c) -regular for constants $a \geq 1$, $b > 1$, and $0 \leq c \leq 1$, if, for problem size N , its I/O complexity satisfies the following recurrence: $Q(N) = a \cdot Q(N/b) + \Theta(1 + N^c/B)$.

Specifically, the algorithm has a recursive calls on sub-problems of size N/b and $\Theta(1)$ **linear/sequential scans** before, between, or after the recursive calls, where the size of the largest scan is $\Theta(N^c)$.

The worst-case performance gap between obliviousness and adaptivity. DAM-optimal (a, b, c) -regular algorithms can be up to an $O(\log N)$ -factor away from being optimally cache-adaptive [7]. Specifically, they are suboptimal in the cache-adaptive model when $a \geq b$ and $c \geq 1$ [7]. Intuitively, (a, b, c) -regular algorithms are not optimal when they contain large sequential reads through memory at each level of the recursion.

The worst-case memory profile. The logarithmic gap in I/O-performance of some (a, b, c) -regular algorithms from being optimally cache-adaptive is based on a worst-case memory workload that adaptively mimics the recursive structure of the algorithm and is tightly synchronized with the execution of the algorithm. Specifically, the worst-case profile provides extra memory to the non-adaptive algorithm during the linear scans (when it cannot benefit from the extra memory) and takes away the extra memory at the end of the linear scans (when it would be able to use the extra memory). The worst-case memory profile is constructed in an adaptive manner by following the execution of the non-adaptive algorithm, MM-SCAN in case of matrix multiplication and EMS in case of sorting.

Example: MM-SCAN and MM-INPLACE. To illustrate the parameter choices for (a, b, c) -regular algorithms, Bender et al. [7] compare two cache-oblivious cubic matrix multiplication algorithms: MM-SCAN and MM-INPLACE [16, 21, 22, 31], and show that only MM-INPLACE is optimally cache adaptive, whereas MM-SCAN is a logarithmic-factor away from being optimally cache adaptive. MM-SCAN divides each input matrix into $b = 4$ blocks and perform $a = 8$ recursive multiplications on the blocks. It uses extra space to store an

intermediate matrix and performs all additions at once with a linear scan. The recurrence relation of MM-SCAN for problem size N (to multiply two matrices of size $\sqrt{N} \times \sqrt{N}$) is $Q(N) = 8Q(N/4) + O(1 + N/B)$. Hence, MM-SCAN has $c = 1$. MM-INPLACE has the same recursive structure, but it avoids the linear scan by performing the multiplications and additions in-place. The recurrence for MM-INPLACE has the same a and b as MM-SCAN, but has $c = 0$: $Q(N) = 8Q(N/4) + O(1)$. Both MM-SCAN and MM-INPLACE are optimal cache-oblivious algorithms, having an I/O cost of $O(N^{3/2}/(\sqrt{M}B))$.

In contrast, MM-INPLACE (with $c < 1$) is optimally cache-adaptive while MM-SCAN (with $c = 1$) is not. This example of MM-SCAN and MM-INPLACE exemplifies how changing the values of a , b , and c can determine whether a cache-oblivious algorithm is optimally cache adaptive or not. Here, the answer depends on whether $c < 1$ or $c = 1$.

From the above discussion, MM-SCAN may seem to be a strictly worse algorithm than MM-INPLACE because MM-SCAN is provably not cache adaptive. However, MM-SCAN is more parallelizable than MM-INPLACE [16] as it can achieve 8-way parallelism whereas MM-INPLACE can only achieve 4-way parallelism.

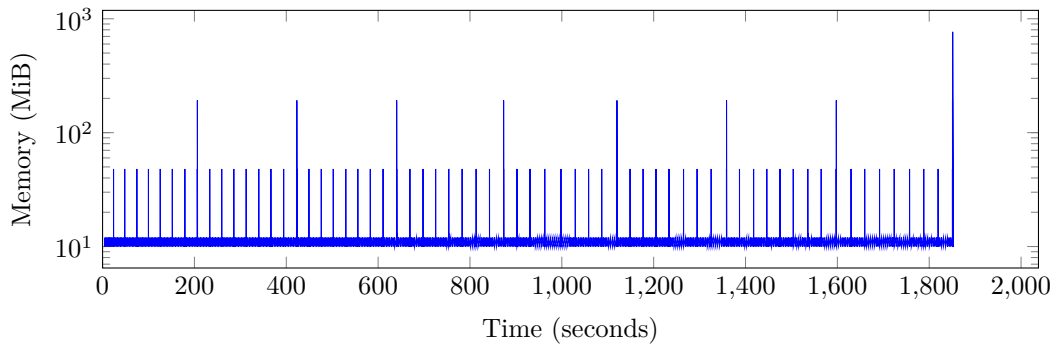
Example: EMS and LFS. Finally, we will review the gap between cache-adaptive and non-adaptive sorting algorithms to see an example of cache-adaptive analysis outside of (a, b, c) -regular algorithms. Both EMS and LFS achieve the disk I/O complexity, $O((N/B) \log_{M/B}(N/B))$ in the external-memory model [1], which is the lower bound for sorting when the cache-parameters remain fixed. However, EMS is not cache-adaptive [7] as it has large linear scans involved and any reduction of memory from the promised amount M during these phases affects the performance of the algorithm heavily. On the other hand, despite not having the same recursive structure as that of the (a, b, c) -regular algorithms, LFS [10, 21] is provably cache-adaptive [8]).

3 Memory profile design

This section presents the high-level design of our evaluation of cache-adaptive and non-adaptive algorithms on a variety of memory profiles for matrix multiplication and external-memory sorting. We test on adaptively constructed memory profiles (e.g. the worst-case profile) as well as “oblivious” memory profiles that are erratic but not adaptively constructed. To perform this evaluation, this study uses a memory profile that does not fluctuate with respect to time. We refer to this as the **constant memory profile** and we use it as a baseline in our experiments.

It is theoretically shown that under **adaptively constructed** worst-case memory workload the performance of two DAM-optimal algorithm diverges. Between two (a, b, c) -regular algorithms, MM-INPLACE is cache adaptive and MM-SCAN is non-adaptive as it is log-factor away from being optimally cache adaptive under this worst-case memory profile. We construct this worst-case **adversarial memory profile** by adapting to the execution of the non-adaptive algorithm and allowing it enough memory to store each recursive sub-problem during its linear scan, and low memory when the algorithm is not performing a linear scan. Since linear scans do not employ any locality of reference, the algorithm only requires $O(1)$ memory; any memory given more than that is not utilized during linear scan. For example, MM-SCAN contains a linear scan at the end of each recursion; in each linear scan of size $N \times N$, memory is increased to $5N^2$ to hold the input and output matrices as well as the intermediate results. However, the cache-adaptive algorithm opportunistically benefits from

16:8 When Are Cache-Oblivious Algorithms Cache Adaptive?



■ **Figure 2** Sample adversarial memory profile generated from running MM-SCAN to multiply two square matrices of width **8192**. The structure of the memory profile mimics the recursive structure of MM-SCAN.

these memory increases, causing its I/O-performance to improve considerably relative to that of the non-adaptive algorithm. Figure 2 shows a sample adversarial memory profile for matrix multiplication.

Similarly, we construct another category of adaptive memory profiles, i.e., the **benevolent memory profile**, but these are designed to be benevolent to the non-adaptive algorithm. The non-adaptive algorithms are unable to use any extra memory that adversarial memory provides during the recursive sequential scans. In the case of the benevolent memory profile, during the sequential scans, instead of increasing the memory we adaptively decrease memory. When the linear scan ends, the memory is again increased to the initial value.

Both the adversarial and benevolent memory profiles are tightly synchronized with the execution of the non-adaptive algorithm. In fact, the adversarial memory profile mimics the recursive structure of the non-adaptive algorithm to bring out the worst performance relative to the adaptive algorithm. In contrast, the benevolent profile aims to minimize the advantage of the cache-adaptive algorithm over the non-adaptive algorithm. However, adversarial and benevolent memory profiles might be too pessimistic or too optimistic in terms of the relative performance of the non-adaptive algorithm and rarely found in the real-world applications. The beyond-worst-case analysis [6] shows the performance gap between MM-SCAN and MM-INPLACE disappears with sufficient smoothing on the worst-case profile, otherwise, the performance gap remains even under smoothed memory profiles. Hence the question remains, how do the non-adaptive algorithms perform under a range of memory profiles that are not generated in an adaptive manner (i.e. the profiles which do not adapt to the execution of the algorithm)?

To gain a more complete view of algorithm performance under memory fluctuations, we run algorithms under **oblivious memory profiles** that are generated non-synthetically and non-adaptively. These profiles are created independently of a given algorithm's execution. A sample oblivious profile can be obtained by running several programs concurrently in shared memory without limiting any of their memory usages [13, 14]. An evaluation under such oblivious workloads accounts for the fact that in practice, often a running process is forced to share available RAM with other concurrent memory-intensive programs. Hence, these oblivious profiles enable us to observe a more complete view of an algorithm's performance under memory fluctuations in a multi-program environment.

4 Evaluation on constant profiles

This section presents details of the experimental setup and studies the performance of the matrix multiplication and external-memory sorting algorithms described in Section 3 on constant memory profiles. All of the tested algorithms are theoretically optimal when the memory does not fluctuate. The empirical results confirm the theory: the performance gap between the cache-adaptive and non-adaptive algorithms does not grow with the problem size when the memory size is fixed.

Experimental setup. All experiments were conducted on a Dell Precision 5820 with an Intel®Core™i9-9900X 3.50GHz processor, two Samsung 16GB M378A2K43DB1-CTD Dual Rank Memory Modules, and Seagate Barracuda ST2000DM001 Desktop SATA Hard Drive. The operating system used is Linux Mint 19.1, Tessa, with kernel version 4.15.0-88. We used C++14 to implement the algorithms, g++ 9.3.0-17ubuntu1 20.04 as compiler, and bash scripts to run the experiments. We measure performance in disk I/Os using `/proc`. At the start of each experiment, we use the `sync` system call to free page-cache and slab objects. We use Linux control groups (`cgroups`) to limit the memory available to a program. We run all the experiments with 25 trials and average the measured I/Os.

Algorithm implementation descriptions. We implemented the two algorithms for MM-SCAN and MM-INPLACE directly from their description in past work on cache-adaptivity [7]. Section 2 provides a detailed description of the divide-and-conquer recursion of these two algorithms.

We explored two algorithms for sorting: External-memory Merge Sort (EMS) and Lazy Funnel Sort (LFS). We implemented an (M/B) -way external-memory merge sort that is compatible with the memory profile generator and chose $M = 256$ MiB and $B = 8$ MiB. Previous work [11] on engineering external sorting algorithms demonstrate that carefully engineered cache-aware sorting algorithms can perform up to $2\times$ better than cache-oblivious ones when the memory does not fluctuate. We expect our results regarding relative algorithm performance in the face of memory fluctuations to hold for other implementations of EMS and LFS because of the difference in the algorithm structure between the two algorithms. For LFS, we used an implementation from Olsen and Skov [27].

Problem sizes. We evaluate the relative performance of MM-SCAN to MM-INPLACE on the constant memory workloads as a function of input size. For each matrix multiplication experiment, we multiply two square matrices. We increase the input matrix width from 512 to 32768. For all the experiments, we provide a fixed memory of 10 MiB to the programs.

Similarly, we study the relative performance of EMS to LFS on the constant memory workloads. For each sorting experiment, we sort an integer array. We increase the input array length from 67 million to 1 billion. We provide a fixed memory of 256 MiB to the programs.

Results. Figure 1a shows that when memory size is fixed, cache-adaptive MM-INPLACE performs roughly $1.8\times$ better than non-adaptive MM-SCAN for all input sizes, i.e., their performance gap does not grow with problem size. On the other hand, Figure 1b shows that cache-aware non-adaptive EMS incurs less disk I/Os (roughly $0.9\times$ on average) than cache-adaptive LFS. This result shows the cache-oblivious sorting algorithm does not bear a major computational overhead when the memory does not fluctuate.

5 Evaluation on adaptive profiles

This section presents details of how we generate the memory fluctuations and studies the performance of the matrix multiplication and sorting algorithms described in Section 3 on adaptively-generated memory workloads. Specifically, it first introduces the memory profile generator used to construct the adaptive profiles. Using this profile generator, we explore two types of adaptive memory profiles: the adversarial profile, and the benevolent profile. Under adversarial workloads, the cache-adaptive algorithms perform up to $4.5\times$ better than the non-adaptive algorithms with increasing problem size. On the other hand, the benevolent memory profile demonstrates that oblivious algorithms adapt well to non-adversarial memory fluctuations while non-oblivious algorithms suffer. MM-INPLACE is up to $1.5\times$ better than MM-SCAN, but the gap does not grow with the problem size. In contrast, LFS performs increasingly better (up to $3.2\times$) than EMS as the problem size grows.

Memory profile generator. We designed a **memory profile generator** that runs a program under a particular memory workload. To fluctuate the memory available to a program, we first used *cgroups*, however *cgroups* does not allow us to decrease the memory available to a program (that the program already claimed) efficiently while the program is running. To address this, the memory profile generator uses *cgroups* to set an initial memory limit available to a program and runs a *balloon* program concurrently with the program within the *cgroups*. The balloon program uses the memory complement to what we intend the program to use. For example, if we want a program to use memory M until time t and thereafter memory $M + \delta$, and the *cgroups* has memory C , the balloon program uses memory $C - M$ until time t and thereafter it uses memory $C - M - \delta$. The initial memory given to a program remains the same as for the constant memory workload.

Using this memory profile generator, we generate two types of profiles, adversarial and benevolent; see Section 3. We measure algorithm performance using the same problem sizes as in Section 4.

Results for the adversarial memory profile

First, let us turn our attention to the adversarial memory workloads. Figure 1 illustrates that under the adversarial memory workloads, the cache-adaptive algorithms perform better than the non-adaptive algorithms, and the gap increases as the problem sizes grow. MM-INPLACE performs roughly up to $4\times$ better than MM-SCAN and LFS performs roughly up to $4.5\times$ better than EMS.

Time spent in linear scan. The performance gap between the cache-adaptive and non-adaptive algorithms grows with the problem size because the relative time spent in scans by the non-adaptive algorithms (MM-SCAN and EMS) grows with problem size. Table 2 shows that the relative time spent in scans grows with problem size for both MM-SCAN (up to 13%) and EMS (up to 76%). Hence, the performance gap also increases between the adaptive and non-adaptive algorithms with problem size. Cache-adaptive algorithms do not perform any such linear scans and take advantage of the extra memory in the adversarial profile. Cache-adaptive algorithms do not perform any such linear scans and take advantage of the extra memory in the adversarial profile.

■ **Table 2** Percentage of running time that the non-adaptive algorithms spend on linear scans.

<i>Algorithm</i>	<i>Input size</i>	<i>% of runtime</i>
MM-SCAN	1024 × 1024	0.1
MM-SCAN	2048 × 2048	4.7
MM-SCAN	4096 × 4096	13.2
EM Merge Sort	512 MiB	43.6
EM Merge Sort	1 GiB	57.9
EM Merge Sort	2 GiB	76.8

Results for the benevolent memory profile

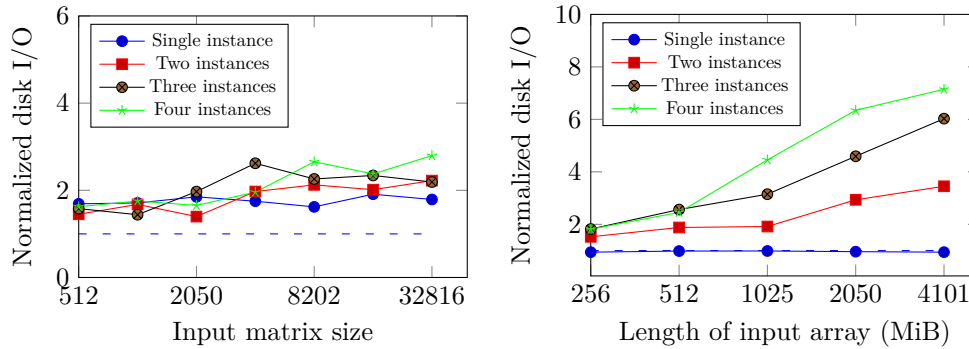
Next, let us turn our attention to the algorithms’ performance under the benevolent memory workloads. Figure 1a shows that MM-SCAN performs very close to MM-INPLACE (between $1.3 \times - 1.6 \times$) on the benevolent profiles since MM-SCAN is not affected by the memory reductions that occurred during the linear scans whereas MM-INPLACE is affected. On the other hand, Figure 1b shows that the gap between EMS and LFS grows with the problem size – EMS incurs up to $3.2 \times$ more I/Os than LFS on the largest input. EMS is unable to adapt to memory reductions during linear scans while LFS adapts to all changes in memory.

6 Evaluation on oblivious profiles

This section explains the setup for the oblivious memory workloads that are not synchronized with the recursive execution of the algorithm and shows that the I/O-performance gap between the two cache-oblivious MM algorithms disappears, but remains between the cache-oblivious and non-oblivious algorithm for sorting. Figure 1a illustrates that MM-SCAN incurs roughly $2 \times$ more disk I/Os on average than MM-INPLACE for oblivious memory profiles as we increase the input size. Figure 1b, on the other hand, shows that EMS incurs increasingly worse than LFS with problem size incurring up to $4.3 \times$ I/Os on average under oblivious memory profiles. In fact, Figure 3b shows EMS may perform up to $7 \times$ more I/Os under oblivious memory workloads.

Experimental setup. This study runs multiple memory-intensive programs concurrently, all sharing the same RAM, to ensure that each program instance experiences memory fluctuations that do not follow the recursive execution of the algorithm. We create these oblivious memory fluctuations in two ways: by creating a **uniform** environment where multiple identically-sized copies of the same program runs concurrently, and by creating a **nonuniform** environment by creating nonuniformity in the concurrent programs. Such nonuniform concurrent programs may include program instances of the same program of different problem sizes, or different programs of same problem sizes. The concurrent programs share a fixed memory that is given at the starting of the experiment. For all the MM experiments, it is 10 MiB, and for all the sorting experiments, it is 256 MiB, if not stated otherwise. We keep the rest of the experimental setup the same as in the case of constant memory profiles (see Section 4).

16:12 When Are Cache-Oblivious Algorithms Cache Adaptive?



(a) Normalized disk I/O of MM-SCAN to MM-INPLACE, instances share a memory of 10 MiB. (b) Normalized disk I/O of EMS to LFS, instances share a memory of 256 MiB.

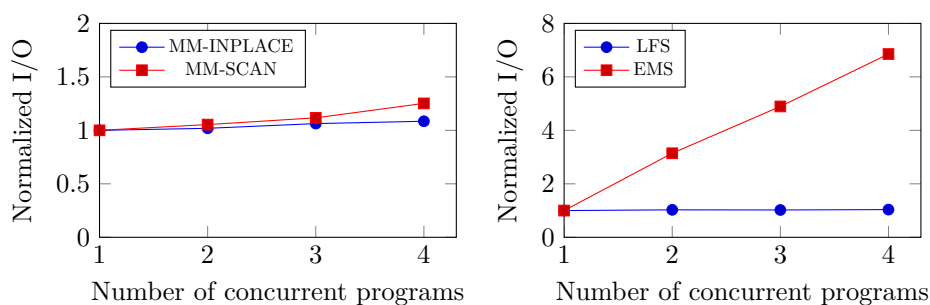
■ **Figure 3** Average disk I/O of a non-adaptive program to the cache-adaptive program as a function of problem size when running up to 4 instances concurrently normalized to the same when a single instance runs concurrently.

Running uniform instances concurrently

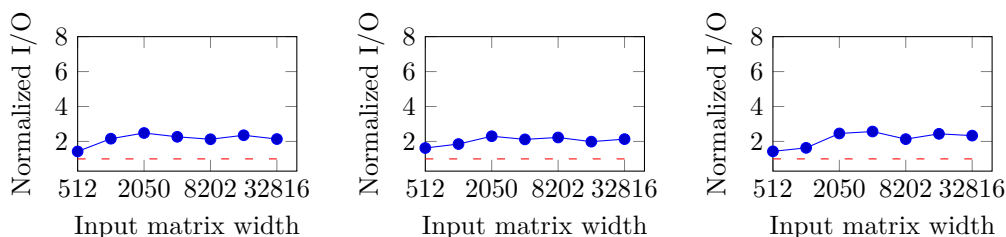
First, let us turn our attention to the oblivious memory workloads that are generated by running multiple concurrent instances of the same program that share a memory [13, 14]. Figure 3a illustrates that cache-oblivious non-adaptive algorithm MM-SCAN performs close to cache-adaptive MM-INPLACE and incurs roughly up to $2\times$ disk I/Os across all problem sizes. On the other hand, Figure 3b shows that the I/O-performance for the cache-aware algorithm EMS rapidly declines relative to the cache-adaptive LFS with increasing problem size ($1.4 - 7.1\times$). Dice et al. [20] showed that multiple threads in shared memory may exhibit a “winner-takes-all” phenomenon where a thread may take memory from the others. However, Figure 3 shows that even if uniformly-sized program instances unevenly share the cache, cache-oblivious algorithms adapt well to memory fluctuations.

Setup. We evaluate the average I/O-performance of the cache-adaptive and non-adaptive algorithms by running concurrent uniform instances, i.e., multiple identically-sized copies of the same program. This multi-program environment setup ensures that each program instance runs under a memory profile that is erratic due to the uncertainty stemming from the concurrent programs’ memory requirement but independent of the recursive structure of the particular program under consideration. We perform the study on uniform instances in two ways. First, we vary the input problem sizes given a number of concurrent program instances. Next, we vary the number of concurrent instances in the range of $1 - 4$ for a given problem size and observe **slowdown**, the degradation of average I/O-performance of a program instance.

Slowdown. Figure 4 illustrates that cache-oblivious algorithms incur relatively small slowdowns even as the number of concurrent instances increases. When we run up to 4 MM-SCAN instances concurrently, the average disk I/Os stays within $1.2\times$ than the disk I/Os incurred by MM-SCAN when a single concurrent instance runs. The MM-INPLACE instances also face almost no slowdown in a similar environment. In contrast, concurrent EMS instances face up to $6\times$ more disk I/Os when compared to an EMS program that does not share memory with other concurrent programs. However, LFS faces a negligible slowdown in the same scenario.



■ **Figure 4** Average normalized disk I/O of a program instance when up to four program instances run concurrently normalized to the disk I/O when a single instance runs.



(a) 3 concurrent instances of the same size. (b) 3 concurrent instances of different sizes. (c) 3 concurrent instances each of MM-SCAN and MM-INPLACE.

■ **Figure 5** Average normalized disk I/O of non-adaptive MM-SCAN to cache-adaptive MM-INPLACE as a function of problem size with 30 MiB memory given. The y-axis indicates that MM-SCAN performs within a constant factor of MM-INPLACE.

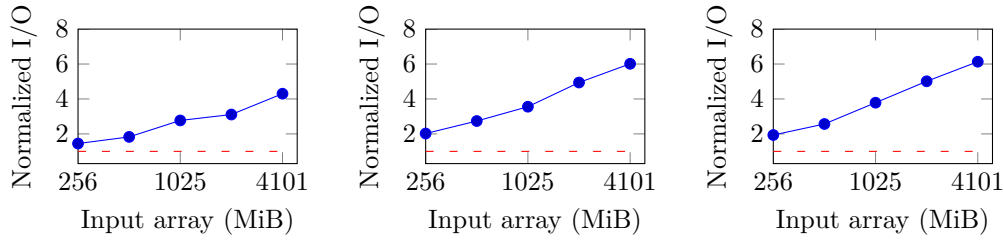
Running nonuniform instances concurrently

Next, let us turn our attention to how the algorithms perform when nonuniform program instances run concurrently and share a fixed memory. The nonuniform programs have different memory requirements at every time step, and therefore are more likely to cause more dramatic memory fluctuations compared to the uniform case. We observe that along with the cache-adaptive algorithms, cache-oblivious MM-SCAN also adapts to the fluctuations caused by nonuniform instances. MM-SCAN incurs up to $2.5\times$ more disk I/Os than MM-INPLACE even when the memory fluctuations are more dramatic in the nonuniform case than in the uniform case, as shown in Figure 5. On the other hand, EMS incurs up to $6\times$ more disk I/Os than LFS, which is more than the gap in the uniform case as demonstrated in Figure 6. Since the memory fluctuations in the nonuniform environment are likely to be more dramatic compared to the uniform environment, the performance gap between EMS and LFS is likely to be more than in the uniform environment because EMS is non-oblivious. Indeed, EMS performs increasingly worse than LFS under the oblivious memory workloads generated in nonuniform environments.

Setup. We create nonuniformity among the program instances in two different manners. In all experiments in Figures 5 and 6, we set the memory size to 30 MiB for MM and to 768 MiB for sorting and retain the problem sizes as mentioned in Section 4.

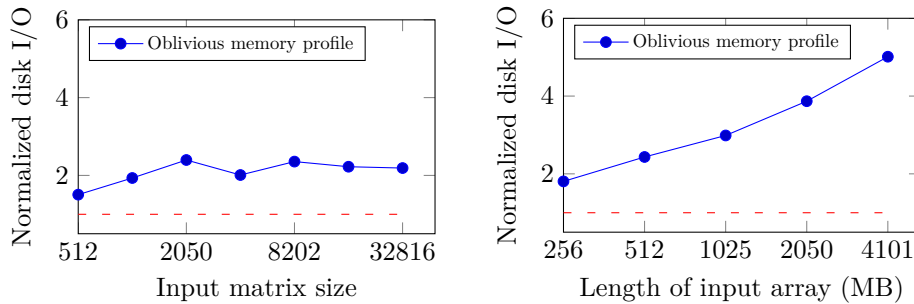
First, we concurrently run 3 instances of the same algorithm on different input sizes. For each experiment, we select 3 input sizes from the available set of sizes uniformly at random. The algorithm in each instance runs sequentially, but the instances run concurrently with

16:14 When Are Cache-Oblivious Algorithms Cache Adaptive?



(a) 3 concurrent instances of the same size. (b) 3 concurrent instances of different sizes. (c) 3 concurrent instances of each of EMS and LFS.

■ **Figure 6** Average normalized disk I/O of non-adaptive EMS to cache-adaptive LFS as a function of problem size with 768 MiB memory given. An increase in the y-axis indicates that EMS performs progressively worse than LFS.



(a) Normalized disk I/O of MM-SCAN to MM-INPLACE.

(b) Normalized disk I/O of EMS to LFS.

■ **Figure 7** Normalized disk I/O of the non-adaptive to the cache-adaptive algorithm for variable problem size for oblivious memory workloads that we generated by running TPC-C concurrently with the programs. An increase in the y-axis means that the non-adaptive algorithm performs progressively worse relative to the adaptive algorithm.

each other. For each experiment, since the smaller instances may complete earlier than the larger ones, we repeat any instances that have finished until all instances have finished. To measure I/O, we report only the first run of each instance per experiment. As a baseline, we also run 3 uniform program instances to evaluate the effect of nonuniform input sizes.

Second, we concurrently run different algorithms for the same problem on the same input size. Specifically, we run 3 instances of the cache-adaptive algorithm (MM-INPLACE or LFS) and 3 instances of the non-adaptive algorithm (MM-SCAN or EMS) concurrently.

Running a database program concurrently

Finally, we turn our attention to how the MM and sorting algorithms perform when run under oblivious memory workloads that we simulated by running a memory-intensive database program concurrently. The database program is not adversarial and therefore not tied to any particular algorithm structure, so we expect the cache-oblivious algorithms to adapt better to the resulting memory fluctuations. Figure 7 confirms this hypothesis – MM-SCAN performs very close to MM-INPLACE (roughly incurring $2.1\times$ disk I/Os), while EMS performs increasingly worse (up to $6\times$) than LFS as the problem size grows.

Setup. As the oblivious memory profiles are generated in a non-synthetic manner (see Section 3), and the memory fluctuations are oblivious to what the algorithm does, we choose to run a concurrent memory-intensive database program with the desired algorithm to evaluate the algorithm’s performance. We generated such an oblivious memory workload by running the TPC-C [17] workload concurrently with either a cache-adaptive or a non-adaptive program and present the normalized disk I/Os of the non-adaptive algorithms to the adaptive algorithms. TPC-C is a popular online transaction processing (OLTP) benchmark used to measure database management systems. Both the MM and sorting algorithms share the memory with the memory-intensive TPC-C program.

7 Conclusion

We investigated how some cache-oblivious and cache-adaptive (and neither) algorithms perform under a wide range of memory profiles. Specifically, we focused on matrix-multiplication algorithms as representatives of (a, b, c) -regular algorithms and sorting algorithms as representatives of cache-optimized but not (a, b, c) -regular algorithms. We experimentally exhibited the gap in I/O-performance between the cache-adaptive and non-adaptive algorithms for MM and sorting under adversarially-generated memory workloads. To do so, we needed to design a profile generator that can dynamically change the amount of available memory depending on a program’s execution. On the other hand, under our oblivious memory workloads, cache-oblivious MM algorithms performed close to each other, whereas the non-oblivious sorting algorithm still performed worse than the cache-oblivious (and adaptive) algorithm. We conjecture that what we have seen in our experiments applies more generally. That is, we conjecture that cache-oblivious programming is a powerful way of empirically achieving cache-adaptivity. This could be important because as mentioned in Section 1, there are many more known cache-oblivious algorithms than known cache-adaptive algorithms. Our results provide hope for the large body of work on cache-oblivious algorithms to empirically perform well even when the cache size fluctuates.

References

- 1 Alok Aggarwal and Jeffrey S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, September 1988.
- 2 Rakesh D. Barve. *Algorithmic Techniques To Overcome The I/O Bottleneck*. PhD thesis, Duke University, 1998.
- 3 Rakesh D. Barve and Jeffrey Scott Vitter. A theoretical framework for memory-adaptive algorithms. In *Proc. 40th Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 273–284, New York City, NY, 1999. IEEE.
- 4 Michael A. Bender, Gerth Stølting Brodal, Rolf Fagerberg, Dongdong Ge, Simai He, Haodong Hu, John Iacono, and Alejandro López-Ortiz. The cost of cache-oblivious searching. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 271–280, Cambridge, MA, 2003. IEEE.
- 5 Michael A. Bender, Gerth Stølting Brodal, Rolf Fagerberg, Dongdong Ge, Simai He, Haodong Hu, John Iacono, and Alejandro López-Ortiz. The cost of cache-oblivious searching. *Algorithmica*, 61(2):463–505, 2011.
- 6 Michael A. Bender, Rezaul A. Chowdhury, Rathish Das, Rob Johnson, William Kuszmaul, Andrea Lincoln, Quanquan C Liu, Jayson Lynch, and Helen Xu. Closing the gap between cache-oblivious and cache-adaptive analysis. In *Proc. 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 63–73, virtual event, USA, 2020. ACM.

16:16 When Are Cache-Oblivious Algorithms Cache Adaptive?

- 7 Michael A. Bender, Erik D. Demaine, Roozbeh Ebrahimi, Jeremy T. Fineman, Rob Johnson, Andrea Lincoln, Jayson Lynch, and Samuel McCauley. Cache-adaptive analysis. In *Proc. 28th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 135–144, Asilomar State Beach, CA, 2016. ACM.
- 8 Michael A. Bender, Roozbeh Ebrahimi, Jeremy T. Fineman, Golnaz Ghasemiasfeh, Rob Johnson, and Samuel McCauley. Cache-adaptive algorithms. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 958–971, Portland, Oregon, 2014. ACM-SIAM.
- 9 Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *Proc. of 41st International Colloquium on Automata, Languages, and Programming*, pages 223–234, Copenhagen, Denmark, 2014. Springer.
- 10 Gerth Stølting Brodal and Rolf Fagerberg. Cache oblivious distribution sweeping. In *Proc. 29th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 426–438, Malaga, Spain, 2002. Springer-Verlag.
- 11 Gerth Stølting Brodal, Rolf Fagerberg, and Kristoffer Vinther. Engineering a cache-oblivious sorting algorithm. *ACM Journal of Experimental Algorithmics*, 12:1–23, 2007.
- 12 Kurt P Brown, Michael James Carey, and Miron Livny. Managing memory to meet multiclass workload response time goals. In *Proc. of the 19th International Conference on Very Large Data Bases (VLDB)*, pages 328–328, Dublin, Ireland, 1993. IEEE.
- 13 Rezaul A. Chowdhury, Pramod Ganapathi, Jesmin Jahan Tithi, Yuan Tang, Charles A. Bachmeier, Bradley C Kuzmaul, Charles E. Leiserson, and Armando Solar Lezama. Autogen: Automatic discovery of efficient recursive divide-&-conquer algorithms for solving dynamic programming problems. In *Proc. of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 1–12, Barcelona, Spain, 2016. ACM.
- 14 Rezaul A. Chowdhury, Pramod Ganapathi, Stephen Tschudi, Jesmin Jahan Tithi, Charles Bachmeier, Charles E. Leiserson, Armando Solar-Lezama, Bradley C. Kuzmaul, and Yuan Tang. Autogen: Automatic discovery of efficient recursive divide-&-conquer algorithms for solving dynamic programming problems. *ACM Transactions on Parallel Computing*, 4(1):1–12, 2017.
- 15 Rezaul Alam Chowdhury and Vijaya Ramachandran. The cache-oblivious gaussian elimination paradigm: theoretical framework, parallelization and experimental evaluation. *Theory of Computing Systems*, 47(4):878–919, 2010.
- 16 T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, Cambridge, Massachusetts, 2001.
- 17 Transaction Processing Performance Council. Tpc-c benchmark. Technical Report 5.10.1, TPC, 2009. URL: <http://www.tpc.org/tpcc/>.
- 18 Peter J. Denning. Thrashing: its causes and prevention. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, AFIPS '68, pages 915–922, San Francisco, CA, 1968. AFIPS.
- 19 Peter J. Denning. Working sets past and present. *IEEE Transactions on Software Engineering*, 6(1):64–84, 1980.
- 20 Dave Dice, Virendra J. Marathe, and Nir Shavit. Brief announcement: Persistent unfairness arising from cache residency imbalance. In *Proc. of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 82–83, Prague, Czech Republic, 2014. ACM.
- 21 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proc. of the 40th Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 285–298, New York City, NY, 1999. IEEE.
- 22 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. *ACM Transactions on Algorithms*, 8(1):4, 2012.
- 23 Masaru Kitsuregawa, Hidehiko Tanaka, and Tohru Moto-Oka. Application of hash to data base machine and its architecture. *New Generation Computing*, 1:63–74, 1983.

- 24 Richard T Mills, Andreas Stathopoulos, and Dimitrios S Nikolopoulos. Adapting to memory pressure from within scientific applications on multiprogrammed cows. In *Proc. 8th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 71–88, Santa Fe, New Mexico, 2004. IEEE.
- 25 Richard T Mills, Chuan Yue, Andreas Stathopoulos, and Dimitrios S Nikolopoulos. Runtime and programming support for memory adaptation in scientific applications via local disk and remote memory. *Journal of Grid Computing*, 5:213–234, 2007.
- 26 Richard Tran Mills. *Dynamic adaptation to CPU and memory load in scientific applications*. PhD thesis, The College of William and Mary, 2004.
- 27 Jesper Holm Olsen, Søren Skov, and Frederik Rønn. Cpp implementations of a cache-oblivious sorting method. Private communication, June 2003.
- 28 J. Ousterhout. Scheduling techniques for concurrent systems. In *Proc. of the 3rd IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 22–30, Miami, Florida, 1982. IEEE.
- 29 HweeHwa Pang, Michael J. Carey, and Miron Livny. Memory-adaptive external sorting. In *Proc. 19th International Conference on Very Large Data Bases (VLDB)*, pages 618–629, Dublin, Ireland, 1993. Morgan Kaufmann.
- 30 HweeHwa Pang, Michael J Carey, and Miron Livny. Partially preemptible hash joins. In *Proc. 5th ACM SIGMOD International Conference on Management of Data (COMAD)*, pages 59–68, Washington, DC, 1993. ACM.
- 31 H. Prokop. Cache oblivious algorithms. Master’s thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1999.
- 32 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis: An attempt to explain the behavior of algorithms in practice. *Communications of the ACM*, 52(10):76–84, 2009.
- 33 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. of the 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 645–654, Las Vegas, NV, USA, 2010. IEEE.
- 34 Weiye Zhang and Per-Åke Larson. A memory-adaptive sort (MASORT) for database systems. In *Proc. 6th International Conference of the Centre for Advanced Studies on Collaborative research (CASCON)*, pages 41–54, Toronto, Ontario, Canada, 1996. IBM Press.
- 35 Weiye Zhang and Per-Åke Larson. Dynamic memory adjustment for external mergesort. In *Proc. 23rd International Conference on Very Large Data Bases (VLDB)*, pages 376–385, Athens, Greece, 1997. Morgan Kaufmann.

Simple Dynamic Spanners with Near-Optimal Recourse Against an Adaptive Adversary

Sayan Bhattacharya ✉

University of Warwick, Coventry, UK

Thatchaphol Saranurak ✉

University of Michigan, Ann Arbor, MI, USA

Pattara Sukprasert ✉

Northwestern University, Evanston, IL, USA

Abstract

Designing dynamic algorithms against an adaptive adversary whose performance match the ones assuming an oblivious adversary is a major research program in the field of dynamic graph algorithms. One of the prominent examples whose oblivious-vs-adaptive gap remains maximally large is the *fully dynamic spanner* problem; there exist algorithms assuming an oblivious adversary with near-optimal size-stretch trade-off using only $\text{polylog}(n)$ update time [Baswana, Khurana, and Sarkar TALG'12; Forster and Goranci STOC'19; Bernstein, Forster, and Henzinger SODA'20], while against an adaptive adversary, even when we allow infinite time and only count recourse (i.e. the number of edge changes per update in the maintained spanner), all previous algorithms with stretch at most $\log^5(n)$ require at least $\Omega(n)$ amortized recourse [Ausiello, Franciosa, and Italiano ESA'05].

In this paper, we completely close this gap with respect to recourse by showing algorithms against an adaptive adversary with near-optimal size-stretch trade-off and recourse. More precisely, for any $k \geq 1$, our algorithm maintains a $(2k - 1)$ -spanner of size $O(n^{1+1/k} \log n)$ with $O(\log n)$ amortized recourse, which is optimal in all parameters up to a $O(\log n)$ factor. As a step toward algorithms with small update time (not just recourse), we show another algorithm that maintains a 3-spanner of size $\tilde{O}(n^{1.5})$ with $\text{polylog}(n)$ amortized recourse *and* simultaneously $\tilde{O}(\sqrt{n})$ worst-case update time.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases Algorithms, Dynamic Algorithms, Spanners, Recourse

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.17

Related Version *Full Version:* [arXiv](#)

Funding Sayan Bhattacharya is supported by Engineering and Physical Sciences Research Council, UK (EPSRC) Grant EP/S03353X/1.

1 Introduction

Increasingly, algorithms are used interactively for data analysis, decision making, and classically as data structures. Often it is not realistic to assume that a user or an adversary is *oblivious* to the outputs of the algorithms; they can be *adaptive* in the sense that their updates and queries to the algorithm may depend on the previous outputs they saw. Unfortunately, many classical algorithms give strong guarantees only when assuming an oblivious adversary. This calls for the design of algorithms that work against an adaptive adversary whose performance match the ones assuming an oblivious adversary. Driven by this question, there have been exciting lines of work across different communities in theoretical computer science, including streaming algorithms against an adaptive adversary [10, 52, 66, 3, 56, 28], statistical algorithms against an adaptive data analyst [51, 37, 7, 63], and very recent algorithms for machine unlearning [47].



© Sayan Bhattacharya, Thatchaphol Saranurak, and Pattara Sukprasert; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 17; pp. 17:1–17:19

Leibniz International Proceedings in Informatics



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

In the area of this paper, namely dynamic graph algorithms, a continuous effort has also been put on designing algorithms against an adaptive adversary. This is witnessed by dynamic algorithms for maintaining spanning forests [53, 60, 67, 61, 34], shortest paths [13, 11, 14, 35, 36, 49, 50, 48, 33], matching [20, 21, 22, 24, 64, 23], and more. This development led to new powerful tools, such as the expander decomposition and hierarchy [62, 42, 59] applicable beyond dynamic algorithms [57, 58, 1, 68], and other exciting applications such as the first almost-linear time algorithms for many flow and cut problems [27, 26, 33, 17]. Nevertheless, for many fundamental dynamic graph problems, including graph sparsifiers [2], reachability [18], directed shortest paths [49], the performance gap between algorithms against an oblivious and adaptive adversary remains large, waiting to be explored and, hopefully, closed.

One of the most prominent dynamic problems whose oblivious-vs-adaptive gap is maximally large is the *fully dynamic spanner* problem [5, 38, 8, 25, 40, 16, 12]. Given an unweighted undirected graph $G = (V, E)$ with n vertices, an α -spanner H is a subgraph of G whose pairwise distances between vertices are preserved up to the *stretch* factor of α , i.e., for all $u, v \in V$, we have $\text{dist}_G(u, v) \leq \text{dist}_H(u, v) \leq \alpha \cdot \text{dist}_G(u, v)$.¹ In this problem, we want to maintain an α -spanner of a graph G while G undergoes both edge insertions and deletions, and for each edge update, spend as small update time as possible.

Assuming an oblivious adversary, near-optimal algorithms have been shown: for every $k \geq 1$, there are algorithms maintaining a $(2k - 1)$ -spanner containing $\tilde{O}(n^{1+1/k})$ edges², which is nearly tight with the $\Omega(n^{1+1/k})$ bound from Erdős' girth conjecture (proven for the cases where $k = 1, 2, 3, 5$ [65]). Their update times are either $O(k \log^2 n)$ amortized [8, 40] or $O(1)^k \log^3 n$ worst-case [16], both of which are polylogarithmic when k is a constant.

In contrast, the only known dynamic spanner algorithm against an adaptive adversary by [5] requires $O(n)$ amortized update time and it can maintain a $(2k - 1)$ -spanner of size $O(n^{1+1/k})$ only for $k \leq 3$. Whether the $O(n)$ bound can be improved remained open until very recently. Bernstein et al. [12] show that a $\log^6(n)$ -spanner can be maintained against an adaptive adversary using $\text{polylog}(n)$ amortized update time. The drawback, however, is that their expander-based technique is too crude to give any stretch smaller than $\text{polylog}(n)$. Hence, for $k \leq \log^6(n)$, it is still unclear if the $\Theta(n)$ bound is inherent. Surprisingly, this holds even if we allow infinite time, and only count *recourse*, i.e., the number of edge changes per update in the maintained spanner. The stark difference in performance between the two adversarial settings motivates the main question of this paper:

Is the $\Omega(n)$ recourse bound inherent for fully dynamic spanners against an adaptive adversary?

Recourse is an important performance measure of dynamic algorithms. There are dynamic settings where changes in solutions are costly while computation itself is considered cheap, and so the main goal is to directly minimize recourse [45, 44, 6, 46]. Even when the final goal is to minimize update time, there are many dynamic algorithms that crucially require the design of subroutines with recourse bounds *stronger than* update time bounds to obtain small final update time [31, 42, 32]. Historically, there are dynamic problems, such as planar embedding [55, 54] and maximal independent set [29, 9, 30], where low recourse algorithms were first discovered and later led to fast update-time algorithms. Similar to dynamic spanners, there are other fundamental problems, including topological sorting [15] and edge coloring [19], for which low recourse algorithms remain the crucial bottleneck to faster update time.

¹ Here, $\text{dist}_G(u, v)$ denotes the distance between u and v in graph G .

² $\tilde{O}(\cdot)$ hides a $\text{polylog}(n)$ factor.

In this paper, we successfully break the $O(n)$ recourse barrier and completely close the oblivious-vs-adaptive gap with respect to recourse for fully dynamic spanners against an adaptive adversary.³

► **Theorem 1.** *There exists a deterministic algorithm that, given an unweighted graph G with n vertices undergoing edge insertions and deletions and a parameter $k \geq 1$, maintains a $(2k - 1)$ -spanner of G containing $O(n^{1+1/k} \log n)$ edges using $O(\log n)$ amortized recourse.*

As the above algorithm is deterministic, it automatically works against an adaptive adversary. Each update can be processed in polynomial time. Both the recourse and stretch-size trade-off of Theorem 1 are optimal up to a $O(\log n)$ factor. When ignoring the update time, it even dominates the current best algorithm assuming an oblivious adversary [8, 40] that maintains a $(2k - 1)$ -spanner of size $O(n^{1+1/k} \log n)$ using $O(k \log^2 n)$ recourse. Therefore, the oblivious-vs-adaptive gap for amortized recourse is closed.

The algorithm of Theorem 1 is as simple as possible. As it turns out, a variant of the classical greedy spanner algorithm [4] simply does the job! Although the argument is short and “obvious” in hindsight, for us, it was very surprising. This is because the greedy algorithm *sequentially* inspects edges in some fixed order, and its output solely depends on this order. Generally, long chains of dependencies in algorithms are notoriously hard to analyze in the dynamic setting. More recently, a similar greedy approach was dynamized in the context of dynamic maximal independent set [9] by choosing a *random order* for the greedy algorithm. Unfortunately, the random order must be kept secret from the adversary and so this fails completely in our adaptive setting. Despite these intuitive difficulties, our key insight is that we can *adaptively choose the order* for the greedy algorithm after each update. This simple idea is enough, see Section 2 for details.

Theorem 1 leaves open the oblivious-vs-adaptive gap for the update time. Below, we show a partial progress on this direction by showing an algorithm with near-optimal recourse and simultaneously non-trivial update time.

► **Theorem 2.** *There exists a randomized algorithm that, given an unweighted graph G with n vertices undergoing edge insertions and deletions, with high probability maintains against an adaptive adversary a 3-spanner of G containing $\tilde{O}(n^{1.5})$ edges using $\tilde{O}(1)$ amortized recourse and $\tilde{O}(\sqrt{n})$ worst-case update time.*

We note again that, prior to the above result, there was no algorithm against an adaptive adversary with $o(n)$ amortized update time that can maintain a spanner of stretch less than $\log^6(n)$. Theorem 2 shows that for 3-spanners, the update time can be $\tilde{O}(\sqrt{n})$ worst-case, while guaranteeing near-optimal recourse.

We prove Theorem 2 by employing a technique called *proactive resampling*, which was recently introduced in [12] for handling an adaptive adversary. We apply this technique on a modification of a spanner construction by Grossman and Parter [43] from distributed computation community. The modification is small, but seems inherently necessary for bounding the recourse.

To successfully apply proactive resampling, we refine the technique in two ways. First, we present a simple abstraction in terms of a certain load balancing problem that captures the power of proactive resampling. Previously, the technique was presented and applied specifically for the dynamic cut sparsifier problem [12]. But actually, this technique is

³ We wish to mention here that our algorithm works against an adaptive adversary that, in addition to just seeing the algorithm’s output, also has access to the internal randomness of the algorithm.

■ **Table 1** The state of the art of fully dynamic spanner algorithms.

Ref.	Stretch	Size	Recourse	Update Time	Deterministic?
Against an oblivious adversary					
[8]	$2k - 1$	$O(k^8 n^{1+1/k} \log^2 n)$	$O(7^{k/2})$ amortized		rand. oblivious
[8, 40]	$2k - 1$	$O(n^{1+1/k} \log n)$	$O(k \log^2 n)$ amortized		rand. oblivious
[16]	$2k - 1$	$\tilde{O}(n^{1+1/k})$	$O(1)^k \log^3 n$ worst-case		rand. oblivious
Against an adaptive adversary					
[5]	3	$O(n^{1+1/2})$	$O(\Delta)$ amortized		deterministic
	5	$O(n^{1+1/3})$	$O(\Delta)$ amortized		deterministic
[12]	$\tilde{O}(1)$	$\tilde{O}(n)$	$\tilde{O}(1)$ amortized		rand. adaptive
	$n^{o(1)}$	$\tilde{O}(n)$	$n^{o(1)}$ worst-case		deterministic
Ours	$2k - 1$	$O(n^{1+1/k} \log n)$	$O(\log n)$ amortized	$\text{poly}(n)$ worst-case	deterministic
	3	$\tilde{O}(n^{1+1/2})$	$\tilde{O}(1)$ amortized	$\tilde{O}(\sqrt{n})$ worst-case	rand. adaptive
	3	$O(n^{1+1/2})$	$O(\min\{\Delta, \sqrt{n}\} \log n)$ worst-case		deterministic

conceptually simple and quite generic, so our new abstraction will likely facilitate future applications. Our second technical contribution is to generalize and make the proactive resampling technique more flexible. At a very high level, in [12], there is a single parameter about sampling probability that is *fixed* throughout the whole process, and their analysis requires this fact. In our load-balancing abstraction, we need to work with multiple sampling probabilities and, moreover, they change through time. We manage to analyze this generalized process using probabilistic tools about *stochastic domination*, which in turn simplifies the whole analysis.

If a strong recourse bound is not needed, then proactive resampling can be bypassed and the algorithm becomes very simple, deterministic, and has slightly improved bounds as follows.

► **Theorem 3.** *There exists a deterministic algorithm that, given an unweighted graph G with n vertices undergoing edge insertions and deletions, maintains a 3-spanner of G containing $O(n^{1.5})$ edges using $O(\min\{\Delta, \sqrt{n}\} \log n)$ worst-case update time, where Δ is the maximum degree of G .*

Despite its simplicity, the above result improves the update time of the fastest deterministic dynamic 3-spanner algorithm [5] from $O(\Delta)$ amortized to $O(\min\{\Delta, \sqrt{n}\} \log n)$ worst-case. In fact, all previous dynamic spanner algorithms with worst-case update time either assume an oblivious adversary [38, 25, 16] or have a very large stretch of $n^{o(1)}$ [12]. See Table 1 for detailed comparison.

Organization. In Section 2, we give a very short proof of Theorem 1. In Section 3, we prove Theorem 2 assuming a crucial lemma (Lemma 8) needed for bounding the recourse. To prove this lemma, we show a new abstraction for the proactive resampling technique in Section 4 and complete the analysis in Section 5. Our side result, Theorem 3, is based on the the static construction presented in Section 3.1 and its simple proof is given in Section 3.2.

2 Deterministic Spanner with Near-optimal Recourse

Below, we show a decremental algorithm that *handles edge deletions only* with near-optimal recourse. This will imply Theorem 1 by a known reduction formally stated in Lemma 6. To describe our decremental algorithm, let us first recall the classic greedy algorithm.

The Greedy Algorithm. Althöfer et al. [4] showed the following algorithm for computing $(2k - 1)$ -spanners. Given a graph $G = (V, E)$ with n vertices, fix some *order* of edges in E . Then, we inspect each edge one by one according to the order. Initially $E_H = \emptyset$. When we inspect $e = (u, v)$, if $\text{dist}_H(u, v) \geq 2k$, then add e into E_H . Otherwise, ignore it. We have the following:

► **Theorem 4** ([4]). *The greedy algorithm above outputs a $(2k - 1)$ -spanner $H = (V, E_H)$ of G containing $O(n^{1+1/k})$ edges.*

It is widely believed that the greedy algorithm is extremely bad in dynamic setting: an edge update can drastically change the greedy spanner. In contrary, when we allow the order in which greedy scans the edges to be changed adaptively, we can avoid removing spanner edges until it is deleted by the adversary. This key insight leads to optimal recourse. When recourse is the only concern, prior to our work this result was known only for spanners with polylog stretch, which is a much easier problem.

The Decremental Greedy Algorithm. Now we describe our deletion-only algorithm. Let G be an initial graph with m edges and $H = (V, E_H)$ be a $(2k - 1)$ -spanner with $O(n^{1+1/k})$ edges. Suppose an edge $e = (u, v)$ is deleted from the graph G . If $(u, v) \notin E_H$, then we do nothing. Otherwise, we do the following. We first remove e from E_H . Now we look at the only remaining non-spanner edges $E \setminus E_H$, one by one in an arbitrary order. (Note that the order is *adaptively* defined and not fixed through time because it is defined only on $E \setminus E_H$.) When we inspect $(x, y) \in E \setminus E_H$, as in the greedy algorithm, we ask if $\text{dist}_H(x, y) \geq 2k$ and add (x, y) to E_H if and only if it is the case. This completes the description of the algorithm.

Analysis. We start with the most crucial point. We claim that the new output after removing e is as if we run the greedy algorithm that first inspects edges in E_H (the order within E_H is preserved) and later inspects edges in $E \setminus E_H$.

To see the claim, we argue that if the greedy algorithm inspects E_H first, then the whole set E_H must be included, just like E_H remains in the new output. To see this, note that, for each $(x, y) \in E_H$, $\text{dist}_H(x, y) \geq 2k$ when (x, y) was inspected according to some order. But, if we move the whole set E_H to be the prefix of the order (while the order within E_H is preserved), it must still be the case that $\text{dist}_H(x, y) \geq 2k$ when (x, y) is inspected and so e must be added into the spanner by the greedy algorithm.

So our algorithm indeed “simulates” inspecting E_H first, and then it explicitly implements the greedy algorithm on the remaining part $E \setminus E_H$. So we conclude that it simulates the greedy algorithm. Therefore, by Theorem 4, the new output is a $(2k - 1)$ -spanner with $O(n^{1+1/k})$ edges.

The next important point is that, whenever an edge e added into the spanner H , the algorithm never tries to remove e from H . So e remains in H until it is deleted by the adversary. Therefore, the total recourse is $O(m)$. With this, we conclude the following key lemma:

► **Lemma 5.** *Given a graph G with n vertices and m initial edges undergoing only edge deletions, the algorithm above maintains a $(2k - 1)$ -spanner H of G of size $O(n^{1+1/k})$ with $O(m)$ total recourse.*

By plugging Lemma 5 to the fully-dynamic-to-decremental reduction by [8] below, we conclude Theorem 1.

► **Lemma 6** ([8]). *Suppose that for a graph G with n vertices and m initial edges undergoing only edge deletions, there is an algorithm that maintains a $(2k - 1)$ -spanner H of size $O(S(n))$ with $O(F(m))$ total recourse where $F(m) = \Omega(m)$, then there exists an algorithm that maintains a $(2k - 1)$ -spanner H' of size $O(S(n) \log n)$ in a fully dynamic graph with n vertices using $O(F(U \log n))$ total recourse. Here U is the total number of updates, starting from an empty graph.*

3 3-Spanner with Near-optimal Recourse and Fast Update Time

In this section, we prove Theorem 2 by showing an algorithm for maintaining a 3-spanner with small update time *in addition* to having small recourse. We start by explaining a basic static construction and needed data structures in Section 3.1 and show the dynamic algorithm in Section 3.2. Assuming our key lemma (Lemma 8) about proactive resampling, most details here are quite straight forward. Hence, due to space constraint, some proofs are omitted. They will appear in our full version.

Throughout this section, we let $N_G(u) = \{v \in V : (u, v) \in E\}$ denote the set of neighbors of a node $u \in V$ in a graph $G = (V, E)$, and we let $\deg_G(u) = |N_G(u)|$ denote the degree of the node u in the graph G .

3.1 A Static Construction and Basic Data Structures

A Static Construction. We now describe our static algorithm. Though our presentation is different, our algorithm is almost identical to [43]. The only difference is that we do not distinguish small-degree vertices from large-degree vertices.

We first arbitrarily partition V into \sqrt{n} equal-sized *buckets* $V_1, \dots, V_{\sqrt{n}}$. We then construct three sets of edges E_1, E_2, E_3 . For every bucket $V_i, i \in [1, \sqrt{n}]$, we do the following. First, for all $v \in V \setminus V_i$, if $V_i \cap N_G(v)$ is not empty, we choose a neighbor $c_i(v) \in V_i \cap N_G(v)$ and add $(v, c_i(v))$ to E_1 . We call $c_i(v)$ an *i -partner* of v . Next, for every edge $e = (u, v)$, where both $u, v \in V_i$, we add e to E_2 . Lastly, for $u, u' \in V_i$ with overlapping neighborhoods, we pick an arbitrary common neighbor $w_{uu'} \in N_G(u) \cap N_G(u')$ and add $(u, w_{uu'}), (w_{uu'}, u')$ to E_3 . We refer to the node $w_{uu'}$ as the *witness* for the pair u, u' .

► **Claim 7.** The subgraph $H = (V, E_1 \cup E_2 \cup E_3)$ is a 3-spanner of G consisting of at most $O(n\sqrt{n})$ edges.

Proof. We need to show that (1) H is a 3-spanner and (2) $E_H = E_1 \cup E_2 \cup E_3$ has size at most $O(n\sqrt{n})$.

Stretch. Consider an edge $e = (u, v)$ where $u \in V_i, v \in V_j$. We show that H has a path of length at most 3 between u and v . The easy case is when $(u, v) \in E_H$. This gives us a path of one edge. It happens when $u = c_i(v)$ or $v = c_j(u)$, or $i = j$. Suppose $(u, v) \notin E_H$. Consider $v' = c_j(u)$. Since u is a common neighbor between v and v' , $P(v, v')$ is not empty. As $e \notin E_H, u \neq w_{vv'}$. As, the path $u, v', w_{vv'}, v$ has length exactly 3, the stretch part is concluded.

Size. Each vertex u has upto \sqrt{n} partners. Since we have n vertices, $|E_1| = O(n\sqrt{n})$. For E_2 , the graph induced on V_i has at most $O(n)$ edges. Since we have \sqrt{n} buckets, $|E_2| = O(n\sqrt{n})$. For E_3 , $|E_3|$ is bounded by the number of witnesses we need. Since we have $O\sqrt{n}$ buckets, and we have $O(n)$ pairs of vertices within the same bucket, for all buckets, $|E_3|$ must be bounded by $O(n\sqrt{n})$. We conclude the proof by saying that $|E_H| = O(|E_1| + |E_2| + |E_3|) = O(n\sqrt{n})$

◁

Dynamizing the Construction. Notice that it suffices to separately maintain E_1, E_2, E_3 , in order to maintain the above dynamic 3-spanner. Maintaining E_1 and E_2 is straightforward and can be done in a fully-dynamic setting in $O(1)$ worst-case update time. Indeed, if $e = (u, c_i(u)) \in E_1$ is deleted, then we pick a new i -partner $c_i(u) \in V_i \cap N_G(u)$ for u . Maintaining $V_i \cap N_G(u)$ for all u allows us to update $c_i(u)$ efficiently. If $e = (u, u') \in E_2$, where $u, u' \in V_i$, is deleted, then we do nothing.

The remaining task, maintaining E_3 , is the most challenging part of our dynamic algorithm. Before we proceed, let us define a subroutine and a data structure needed to implement our algorithm.

Resampling Subroutine. We define $\text{RESAMPLE}(u, u')$ as a subroutine that *uniformly samples* a witness $w_{uu'}$ (i.e. a common neighbor of u and u'), if exists. Notice that, we can obtain E_3 by calling $\text{RESAMPLE}(u, u')$ for all $u, u' \in V_i$ and for all $i \in [1, \sqrt{n}]$.

■ **Algorithm 1** Maintaining a 3-spanner after one edge deletion.

Data: $G = (V, E)$, $H = (V, E_1 \cup E_2 \cup E_3)$
Input: $e = (u, v)$ is an edge being deleted
Result: $G' = (V, E')$, $H' = (V, E'_1 \cup E'_2, E'_3)$

```

1 begin
2    $E' \leftarrow E \setminus e$ 
3   Let  $i_u$  be such that  $u \in V_{i_u}$ 
4   Let  $i_v$  be such that  $v \in V_{i_v}$ 
5
6   if  $u = c_{i_u}(v)$  then
7     Reselect  $c_{i_u}(v)$  from  $V_{i_u} \cap N_{G'}(v)$ 
8   if  $v = c_{i_v}(u)$  then
9     Reselect  $c_{i_v}(u)$  from  $V_{i_v} \cap N_{G'}(u)$ 
10   $E'_1 \leftarrow E_1 \setminus e \cup \{(u, c_{i_u}(v)), (v, c_{i_v}(u))\}$ 
11
12  if  $i_u = i_v$  then
13     $E'_2 \leftarrow E_2 \setminus e$ 
14
15   $E'_3 \leftarrow E_3 \setminus e$ 
16  for  $u' \in V_{i_u} \setminus u$  do
17    if  $v = w_{uu'}$  then
18       $w_{uu'} \leftarrow \text{RESAMPLE}(u, u')$  /* RESAMPLE( $\cdot, \cdot$ ) is drawn from  $G'$  */
19
20     $E'_3 \leftarrow E'_3 \cup \{(u, w_{uu'}), (w_{uu'}, u')\}$ 
21  for  $v' \in V_{i_v} \setminus v$  do
22    if  $u = w_{vv'}$  then
23       $w_{vv'} \leftarrow \text{RESAMPLE}(v, v')$ 
24     $E'_3 \leftarrow E'_3 \cup \{(v, w_{vv'}), (w_{vv'}, v')\}$ 

```

Partnership Data Structures. The subroutine above hints that we need a data structure for maintaining the common neighborhoods for all pairs of vertices that are in the same bucket. For vertices u and v within the same bucket, we let $P(u, v) = N_G(u) \cap N_G(v)$ be

the *partnership* between u and v . To maintain these structures dynamically, when an edge (u, v) is inserted, if $u \in V_i$ and $v \in V_j$, we add v to $P(u, u')$ for all $u' \in V_i \cap N_G(v) \setminus \{u\}$, and symmetrically add u to $P(v, v')$ for all $v' \in V_j \cap N_G(u) \setminus \{v\}$. This clearly takes $O(\sqrt{n} \log n)$ worst-case time for edge insertion, and this is symmetric for edge deletion.

As we want to prove that our final update time is $\tilde{O}(\sqrt{n})$, we can assume from now that E_1, E_2 , and all partnerships are maintained in the background. The pseudocode for maintaining the 3-spanner is provided in Algorithm 1.

3.2 Maintaining Witnesses via Proactive Resampling

Remark. For clarity of exposition, we will present an amortized update time analysis. Using standard approach, we can make the update time worst case. We will discuss this issue at the end of this section.

Our dynamic algorithm runs in *phases*, where each phase lasts for $n\sqrt{n}$ consecutive updates (edge insertions/deletions). As a spanner is decomposable⁴, it suffices to maintain a 3-spanner H of the graph undergoing only edge deletions within this phase and then include all edges inserted within this phase into H , which increases the size of H by at most $n\sqrt{n}$ edges. Henceforth, *we only need to present how to initialize a phase and how to handle edge deletions within each phase*. The reason behind this reduction is because our proactive resampling technique naturally works for decremental graphs.

Initialization. At the start of the phase, since our partnerships structures only processed edge deletions from the previous phase, we first update partnerships with all the $O(n\sqrt{n})$ inserted edges from the previous phase. Then, we call $\text{RESAMPLE}(u, u')$ for all $u, u' \in V_i$ for all $i \in [1, \sqrt{n}]$ to replace all witnesses and initialize E_3 of this phase.

Difficulty of Bounding Recourse. Maintaining E_3 (equivalently, the witnesses) in $\tilde{O}(\sqrt{n})$ worst-case time is straightforward because the partnership data structure has $O(\sqrt{n} \log n)$ update time. However, our goal is to show $\tilde{O}(1)$ amortized recourse, which is the most challenging part of our analysis. To see the difficulty, if (u, v) is deleted and $u \in V_i$, a vertex v may serve as a witness $\{w_{uu'}\}$ for all $u' \in V_i$. In this case, deleting (u, v) causes the algorithm to find a new witness $w_{uu'}$ for all $u' \in V_i$. This implies a recourse of $|V_i| = \Omega(\sqrt{n})$. To circumvent this issue, we apply the technique of *proactive resampling*, as described below.

Proactive Resampling. We keep track of a *time-variable* T ; the number of updates to G that have occurred in this phase until now. T is initially 0. We increment T each time an edge gets deleted from G .

In addition, for all $i \in [1, \sqrt{n}]$ and $u, u' \in V_i$ with $u \neq u'$, we maintain: (1) $w_{uu'}$, the *witness* for the pair u and u' and (2) a set $\text{SCHEDULE}[u, u']$ of positive integers, which is the set of timesteps where our algorithm intends to *proactively resample* a new witness for u, u' . This set grows adaptively each time the adversary deletes $(u, w_{uu'})$ or $(w_{uu'}, u')$.

Finally, to ensure that the update time of our algorithm remains small, for each $\lambda \in [1, n\sqrt{n}]$ we maintain a $\text{LIST}[\lambda]$, which consists of all those pairs of nodes (x, x') such that $\lambda \in \text{SCHEDULE}[x, x']$.

⁴ Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$. If H_1 and H_2 are α -spanners G_1 and G_2 respectively, then $H_1 \cup H_2$ is a α -spanner of $G_1 \cup G_2$.

When an edge (u, v) , where $u \in V_i$ and $v \in V_j$ is deleted, we do the following operations. First, for all $u' \in V_i$ that had $v = w_{uu'}$ as a common neighbor with u before deleting (u, v) , we add the timesteps $\{T + 2^k \mid T + 2^k \leq n\sqrt{n}, k \in \mathbb{N}\}$ to $\text{SCHEDULE}[u, u']$. Second, analogous to the previous one, for all $v' \in V_j$ that had $u = w_{vv'}$ as a common neighbor with v before deleting (u, v) , we add the timesteps $\{T + 2^k \mid T + 2^k \leq n\sqrt{n}, k \in \mathbb{N}\}$ to $\text{SCHEDULE}[v, v']$. Third, we set $T \leftarrow T + 1$. Lastly, for each $(x, x') \in \text{LIST}[T]$, we call the subroutine $\text{RESAMPLE}(x, x')$.

The key lemma below summarizes a crucial property of this dynamic algorithm. Its proof appears in Section 4.

► **Lemma 8.** *During a phase consisting of $n\sqrt{n}$ edge deletions, our dynamic algorithm makes at most $\tilde{O}(\sqrt{n})$ calls to the RESAMPLE subroutine after each edge deletion. Moreover, the total number of calls to the RESAMPLE subroutine during an entire phase is at most $\tilde{O}(n\sqrt{n})$ w.h.p. Both these guarantees hold against an adaptive adversary.*

Analysis of Recourse and Update Time. Our analysis are given in the lemmas below.

► **Lemma 9 (Recourse).** *The amortized recourse of our algorithm is $\tilde{O}(1)$ w.h.p., against an adaptive adversary.*

Proof. To maintain the edge-sets E_1 and E_2 , we pay a worst-case recourse of $O(1)$ per update. For maintaining the edge-set E_3 , our total recourse during the entire phase is at most $O(1)$ times the number of calls made to the $\text{RESAMPLE}(\cdot, \cdot)$ subroutine, which in turn is at most $\tilde{O}(n\sqrt{n})$ w.h.p. (see Lemma 8). Finally, while computing E_3 in the beginning of a phase, we pay $O(n\sqrt{n})$ recourse. Therefore, the overall total recourse during an entire phase is $\tilde{O}(n\sqrt{n})$ w.h.p.. Since a phase lasts for $n\sqrt{n}$ time steps, we conclude the lemma. ◀

► **Lemma 10 (Worst-case Update Time within a Phase).** *Within a phase, our algorithm handles a given update in $\tilde{O}(\sqrt{n})$ worst case time w.h.p..*

Proof. Recall that the sets E_1, E_2 can be maintained in $O(1)$ worst case update time. Henceforth, we focus on the time required to maintain the edge-set E_3 after a given update in G .

Excluding the time spent on maintaining the partnership data structure (which is $\tilde{O}(\sqrt{n})$ in the worst-case anyway), this is proportional to $\tilde{O}(1)$ times the number of calls made to the $\text{RESAMPLE}(\cdot, \cdot)$ subroutine, plus $\tilde{O}(1)$ times the number of pairs $u, u'(v, v')$ where we need to adjust $\text{SCHEDULE}[u, u']$. The former is w.h.p. at most $\tilde{O}(\sqrt{n})$ according to Lemma 8, while the latter is also at most $\tilde{O}(\sqrt{n})$ since $|V_i|, |V_j| \leq \sqrt{n}$. Thus, within a phase we can also maintain the set E_3 w.h.p. in $\tilde{O}(\sqrt{n})$ worst case update time. ◀

Although the above lemma says that we can handle each edge deletion in $\tilde{O}(\sqrt{n})$ worst-case update time, our current algorithm does not guarantee worst-case update time yet because the initialization time exceed the $\tilde{O}(\sqrt{n})$ bound. In more details, observe that the total initialization time is $O(n\sqrt{n}) \times O(\sqrt{n} \log n)$ because we need to insert $O(n\sqrt{n})$ edges into partnership data structures, which has $O(\sqrt{n} \log n)$ update time. Over a phase of $n\sqrt{n}$ steps, this implies only $\tilde{O}(\sqrt{n})$ amortized update time.

However, since the algorithm takes long time only at the initialization of the phase, but takes $\tilde{O}(\sqrt{n})$ worst-case step for each update during the phase, we can apply the standard building-in-the-background technique to de-amortized the update time.⁵ We conclude the following:

⁵ The proof for this standard technique will be included in our full version.

► **Lemma 11** (Worst-case Update Time for the Whole Update Sequence). *W.h.p., the worst-case update time of our dynamic algorithm is $\tilde{O}(\sqrt{n})$.*

4 Proactive Resampling: Abstraction

The goal of this section is to prove Lemma 8 for bounding the recourse of our 3-spanner algorithm. This is the most technical part of this paper. To ease the analysis, we will abstract the problem situation in Lemma 8 as a particular dynamic problem of assigning jobs to machines while an adversary keeps deleting machines and the goal is to minimize the total number of reassignments. Below, we formalize this problem and show how to use it to bound the recourse of our 3-spanner algorithm.

Our abstraction has two technical contributions: (1) it allows us to easily work with multiple sampling probabilities, while in [12], they fixed a single parameter on sampling probability, (2) the simplicity of this abstraction can expose the generality of the proactive resampling technique itself; it is not specific to the cut sparsifier problem as used in [12].

Jobs, Machines, Routines, Assignments, and Loads. Let J denote a set of *jobs* and M denote a set of *machines*. We think of them as two sets of vertices of the (hyper)-graph $G = (J, M, R)$.⁶ A *routine* $r \in R$ is a hyperedge of G such that r contains exactly one job-vertex from J , denoted by $\text{job}(r) \in J$, and may contain several machine-vertices from M , denoted by $M(r) \subseteq M$. Each routine r in G means there is a routine for handling $\text{job}(r)$ by *simultaneously* calling machines in $M(r)$. Note that $r = \{\text{job}(r)\} \cup M(r)$. We say that r is a *routine for* $\text{job}(r)$. For each machine $x \in M(r)$, we say that routine r *involves* machine x , or that r *contains* x . The set $R(x)$ is then defined as the set of routines involving machine x . Observe that there are $\deg_G(u)$ different routines for handling job u . An *assignment* $A = (J, M, F \subseteq R)$ is simply a subgraph of G . We say assignment A is *feasible* iff $\deg_A(u) = 1$ for every job $u \in J$ where $\deg_G(u) > 0$. That is, every job is handled by some routine, if exists. When $r \in A$, we say that $\text{job}(r)$ is *handled by* routine r . Finally, given an assignment A , the *load* of a machine x is the number of routines in A involving x , or in other words, is the degree of x in A , $\deg_A(x)$. We note explicitly that our end-goal is not to optimize loads of machines. Rather, we want to minimize the number of reassignments needed to maintain feasible assignments throughout the process.

In this section, we usually use u, v, w to denote jobs, use x, y, z to denote machines, and use e or r to denote routines or (hyper)edges.

The Dynamic Problem. Our problem is to maintain a feasible assignment A while the graph G undergoes a sequence of machine deletions (which might stop before all machines are deleted). More specifically, when a machine x is deleted, all routines r containing x are deleted as well. But when routines in A are deleted, A might not be feasible anymore and we need to add new edges to A to make A feasible. Our goal is to minimize the total number of routines ever added to A .

To be more precise, write the graph G and the assignment A after t machine-deletions as $G^t = (J, M, R^t)$ and $A^t = (J, M, F^t)$, respectively. Here, we define *recourse* at timestep t to be $|F^t \setminus F^{t-1}|$, which is the number of routines added into A at timestep t . When the adversary deletes T machines, the goal is then to minimize the total recourse $\sum_{t=1}^T |F^t \setminus F^{t-1}|$.

⁶ This graph is different from the graph that we maintain a spanner in previous sections.

The Algorithm: Proactive Resampling. To describe our algorithm, first let $\text{RESAMPLE}(u)$ denote the process of reassigning job u to a uniformly random routine for u . In the graph language, $\text{RESAMPLE}(u)$ removes the edge r such that $\text{job}(r) = u$ from A , sample an edge r' from $\{r \in R \mid \text{job}(r) = u\}$, and then add r' into A . At timestep 0, we initialize a feasible assignment A^0 by calling $\text{RESAMPLE}(u)$ for every job $u \in J$, i.e., assign each job u to a random routine for u . Below, we describe how to handling deletions.

Let T be the total number of machine-deletions. For each job u , we maintain $\text{SCHEDULE}(u) \subseteq [T]$ containing all time steps that we will invoke $\text{RESAMPLE}(u)$. That is, at any timestep t , before an adversary takes any action, we call $\text{RESAMPLE}(u)$ if $t \in \text{SCHEDULE}(u)$.

We say that an adversary *touches* u at timestep t if the routine $r \in A^t$ handling u at time t is deleted. When u is touched, we call $\text{RESAMPLE}(u)$ and, very importantly, we put $t + 1, t + 2, t + 4, \dots$ where $t + 2^i \leq T$ into $\text{SCHEDULE}(u)$. This is the action that we call *proactive resampling* because we do not just resample a routine for u only when u is touched, but do so proactively in the future as well. This completes the description of the algorithm.

Clearly, A remains a feasible assignment throughout because whenever a job u is touched, we immediately call $\text{RESAMPLE}(u)$. The key lemma below states that the algorithm has low recourse, even if the adversary is adaptive in the sense that each deletion at time t depends on previous assignment before time t .

► **Lemma 12.** *Let T be the total number of machine-deletions. The total recourse of the algorithm running against an adaptive adversary is $O(|J| \log(\Delta) \log^2 |M| + T \log^2 |M|)$ with high probability where Δ is the maximum degree of any job. Moreover, if the load of a machine never exceeds λ , then our algorithm has $O(\lambda \log T)$ worst-case recourse.*

We will prove Lemma 12 in Section 5. Before proceeding any further, however, we argue why Lemma 12 directly bounds the recourse of our 3-spanner algorithm.

Back to 3-spanners: Proof of Lemma 8. It is easy to see that maintaining E_3 in our 3-spanner algorithm can be framed exactly as the job-machine load-balancing problem. Suppose the given graph is $G = (V, E)$ where $n = |V|$ and $m = |E|$. We create a job $j_{uu'}$ for each pair of vertices $u, u' \in V_i$ with $u \neq u'$. For each edge $e \in E$, we create a machine x_e . Hence, $|J| = O(n^{1.5})$ and $|M| = |E| = m$. For each job, as we want to have a witness $w_{uu'}$, this witness is corresponding to two edges $e = (u, w_{uu'})$ and $e' = (u', w_{uu'})$. Hence, we create a routine $r = (j_{uu'}, e, e')$ for each $u, u' \in V_i$ and a common neighbor $w_{uu'}$. Since there are at most n common neighbors between each u and u' , $\Delta = O(n)$. A feasible assignment is then corresponding to finding a witness for each job. Our algorithm that maintains the spanner is also exactly this load-balancing algorithm. Hence, the recourse of the 3-spanner construction follows from Lemma 12 where we delete exactly $T = O(n^{1.5})$ machines. As $|J| = O(n^{1.5})$, the total recourse bound then becomes $O(n^{1.5} \log^3 n)$. As $T = O(n^{1.5})$, averaging this bound over all timesteps yields $O(\log^3 n)$ amortized recourse.

5 Proactive Resampling: Analysis (Proof of Lemma 12)

The first step to prove Lemma 12 is to bound the loads of machines x . This is because whenever machine x is deleted, its load of $\deg_A(x)$ would contribute to the total recourse.

What would be the expected load of each machine? For intuition, suppose that the adversary was *oblivious*. Recall that $R(x)$ denote the set of all routines involving machine x . Then, the expected load of machine x would be $\sum_{r \in R(x)} 1 / \deg_G(\text{job}(r))$ because each

17:12 Simple Dynamic Spanners with Near-Optimal Recourse

job samples its routine uniformly at random, and this is concentrated with high probability using Chernoff's bound. Although in reality our adversary is *adaptive*, our plan is to still prove that the loads of machines do not exceed its expectation in the oblivious setting too much. This motivates the following definitions.

► **Definition 13.** *The target load of machine x is $\text{target}(x) = \sum_{r \in R(x)} 1/\text{deg}_G(\text{job}(r))$. The target load of x at time t is $\text{target}^t(x) = \sum_{r \in R^t(x)} 1/\text{deg}_{G^t}(\text{job}(r))$. An assignment A has overhead (α, β) iff $\text{deg}_A(x) \leq \alpha \cdot \text{target}(x) + \beta$ for every machine $x \in M$.*

Our key technical lemma is to show that, via proactive resampling, the loads of machines are indeed close to its expectation in the oblivious setting. That is, the maintained assignment has small overhead. Recall that T is the total number of machine-deletions.

► **Lemma 14.** *With high probability, the assignment A maintained by our algorithm always has overhead $(O(\log T), O(\log |M|))$ even when the adversary is adaptive.*

We defer the proof of Lemma 14 to Section 6. Assuming Lemma 14, we formally show how to bound of machine loads implies the total recourse, which proves Lemma 12.

Proof of Lemma 12. Let T be the total number of deletions. Observe that the total recourse up to time T is precisely the total number of `RESAMPLE(.)` calls up to time T , which in turn is at most the total number of `RESAMPLE(.)` calls put into `SCHEDULE(.)` since time 1 until time T . Therefore, our strategy is to bound, for each time t , the number of `RESAMPLE(.)` calls *newly generated* at time t . Let x^t be the machine deleted at time t . Observe this is at most $O(\log T) \times \text{deg}_{A^t}(x^t)$ where $\text{deg}_{A^t}(x^t)$ is x^t 's load at time t and the $O(\log T)$ factor is because of proactive sampling.

By Lemma 14, we have $\text{deg}_{A^t}(x^t) \leq O(\log(T)\text{target}^t(x^t) + \log |M|)$. Also, we claim that $\sum_{t=1}^T \text{target}^t(x^t) = O(|J| \log \Delta)$ where Δ is the maximum degree of jobs (to be proven below). Therefore, the total recourse up to time T is at most

$$\begin{aligned} O(\log T) \sum_{t=1}^T \text{deg}_{A^t}(x^t) &\leq O(\log T) \sum_{t=1}^T O(\log(T)\text{target}^t(x^t) + \log |M|) \\ &\leq O(|J| \log(\Delta) \log^2 |M| + T \log^2 |M|) \end{aligned}$$

as $T \leq |M|$.

It remains to show that $\sum_{t=1}^T \text{target}^t(x^t) = O(|J| \log \Delta)$. Recall that $\text{target}^t(x) = \sum_{r \ni x} \frac{1}{\text{deg}_{G^t}(\text{job}(r))}$. Imagine when machine x^t is deleted at time t . We will show how to charge $\text{target}^t(x^t)$ to jobs with edges connecting to x^t . For each job u with c (hyper)edges connecting to x^t , u 's contribution of $\text{target}^t(x^t)$ is $c/\text{deg}_{G^t}(u)$. So we distribute the charge of $c/\text{deg}_{G^t}(u) \leq \frac{1}{\text{deg}_{G^t}(u)} + \frac{1}{\text{deg}_{G^t}(u)-1} + \dots + \frac{1}{\text{deg}_{G^t}(u)-c+1}$ to u . Since these edges are charged from machine x^t to job u only once, the total charge of each job u at most $\frac{1}{\text{deg}_G(u)} + \frac{1}{\text{deg}_G(u)-1} + \dots + 1/2 + 1 = O(\log \Delta)$. Since there are $|J|$ jobs, the bound $\sum_{t=1}^T \text{target}^t(x^t) = O(|J| \log \Delta)$ follows.

To see that we have worst-case recourse, one can look at any timestep t . There are $O(\log t)$ timesteps that can cause `RESAMPLE` to be invoked at timestep t , namely, $t-1, t-2, t-4, \dots$. At each of these timesteps t' , one machine is deleted, so the number of `RESAMPLE` calls added from timestep t' is also bounded by the load of the deleted machine $x_{t'}$, which does not exceed λ . Summing this up, the number of calls we make at timestep t is at most $O(\lambda \log t) = O(\lambda \log T)$. This concludes our proof. ◀

6 Proof of Lemma 14

Here, we show that the load $\deg_{A^t}(x)$ of every machine x at each time t is small. Some basic notions are needed in this analysis.

Experiments and Relevant Experiments. An *experiment* X is a binary random variable associated with an edge/routine e and time step t , where $X = 1$ iff $\text{RESAMPLE}(\text{job}(e))$ is called at time t and e is chosen to handle $\text{job}(e)$, among all edges incident to $\text{job}(e)$. Observe that $\mathbb{P}[X = 1] = 1/\deg_{G^t}(\text{job}(e))$. Note that each call to $\text{RESAMPLE}(u)$ at time t creates new $\deg_{G^t}(u)$ experiments. We order all experiments X_1, X_2, X_3, \dots by the time of their creation. For convenience, for each experiment X , we let $e(X)$, $t(X)$, and $\text{job}(X)$ denote its edge, time of creation, and job respectively.

Next, we define the most important notion in the whole analysis.

► **Definition 15.** For any time t and edge $e \in R^t$ at time t , an experiment X is (t, e) -relevant if

- $e(X) = e$, and
- there is no $t(X) < t' < t$ such that $t' \in \text{SCHEDULE}^{t(X)}(\text{job}(e))$.

Moreover, X is (t, x) -relevant if it is (t, e) -relevant and edge $e \in R^t(x)$ is incident to x .

Intuitively, X is a (t, e) -relevant experiment if X could cause e to appear in the assignment A^t at time t . To see why, clearly if $e(X) \neq e$, then X cannot cause e to appear. Otherwise, if $e(X) = e$ but there is $t' \in (t(X), t)$ where $t' \in \text{SCHEDULE}^{t(X)}(\text{job}(e))$, then X cannot cause e to appear at time t either. This is because even if X is successful and so e appears at time $t(X)$, then later at time $t' > t(X)$, e will be resampled again, and so X has nothing to do whether e appears at time $t > t'$. With the same intuition, X is (t, x) -relevant if X could contribute to the load $\deg_{A^t}(x)$ of machine x at time t .

Notice that we decide whether X is a (t, e) -relevant based on $\text{SCHEDULE}^{t(X)}(\text{job}(e))$ at time $t(X)$. If it was based on $\text{SCHEDULE}^t(\text{job}(e))$ at time t , then there would be only a single experiment X that is (t, e) -relevant (which is the one with $e(X) = e$ and maximum $t(X) < t$).

According to Definition 15, there could be more than one experiments that are (t, e) -relevant. For example, suppose X is (t, e) -relevant. At time $t(X) + 1$, the adversary could touch $\text{job}(e)$, hence, adding $t(X) + 2, t(X) + 4, \dots$ into $\text{SCHEDULE}(\text{job}(e))$. Because of this action, there is another experiment X' that is (t, e) -relevant and $t(X') > t(X)$. This motivates the following definition.

► **Definition 16.** Let $\text{Rel}(t, e)$ be the random variable denoting the number of (t, e) -relevant experiments, and let $\text{Rel}(t, x) = \sum_{e \in R^t(x)} \text{Rel}(t, e)$ denote the total number of (t, x) -relevant experiments.

To simplify the notations in the proof of Lemma 14 below, we assume the following.

► **Assumption 17 (The Machine-disjoint Assumption).** For any routines e, e' with $\text{job}(e) = \text{job}(e')$, then $M(e) \cap M(e') = \emptyset$. That is, the edges adjacent to the same job are machine-disjoint.

Note that this assumption indeed holds for our 3-spanner application. This is because any two paths of length 2 between a pair of centers u and u' must be edge disjoint in any simple graph. We show how remove this assumption in our full version. The notations there will slightly be more complicated.

17:14 Simple Dynamic Spanners with Near-Optimal Recourse

Roadmap for Bounding Loads. We are now ready to describe the key steps for bounding the load $\deg_{A^t}(x)$, for any time t and machine x .

First, we write $\mathcal{X}^{(t,x)} = X_1^{(t,x)}, X_2^{(t,x)}, \dots, X_{\text{Rel}(t,x)}^{(t,x)}$ as the sequence of all (t,x) -relevant experiments (ordered by time step the experiments are taken). The order in $\mathcal{X}^{(t,x)}$ will be important only later. For now, we write

$$\overline{\deg}_{A^t}(x) = \sum_{X \in \mathcal{X}^{(t,x)}} X,$$

as the total number of success (t,x) -relevant experiments. As any edge e adjacent to x in A^t may appear only because of some successful (t,x) -relevant experiment $X \in \mathcal{X}^{(t,x)}$, we conclude the following:

► **Lemma 18** (Key Step 1). $\deg_{A^t}(x) \leq \overline{\deg}_{A^t}(x)$.

Lemma 18 reduces the problem to bounding $\overline{\deg}_{A^t}(x)$. If all (t,x) -relevant experiments $\mathcal{X}^{(t,x)} = \{X_i^{(t,x)}\}_i$ were independent, then we could have easily applied standard concentration bounds to $\overline{\deg}_{A^t}(x) = \sum_{X \in \mathcal{X}^{(t,x)}} X$. Unfortunately, they are not independent as the outcome of earlier experiments can affect the adversary's actions, which in turn affect later experiments.

Our strategy is to relate the sequence $\mathcal{X}^{(t,x)}$ of (t,x) -relevant experiments to another sequence $\hat{\mathcal{X}}^{(t,x)} = \hat{X}_1^{(t,x)}, \hat{X}_2^{(t,x)}, \dots, \hat{X}_{\text{Rel}(t,x)}^{(t,x)}$ of *independent* random variables defined as follows. For each (t,x) -relevant experiment $\hat{X}_i^{(t,x)}$ where $e = e(\hat{X}_i^{(t,x)})$ and $u = \text{job}(e)$, we carefully define $\hat{X}_i^{(t,x)}$ as an *independent* binary random variable such that

$$\mathbb{P}[\hat{X}_i^{(t,x)} = 1] = 1/\deg_{G^t}(u),$$

which is the probability that $\text{RESAMPLE}(u)$ chooses e at time t . We similarly define

$$\widehat{\deg}_{A^t}(x) = \sum_{\hat{X} \in \hat{\mathcal{X}}^{(t,x)}} \hat{X},$$

that sums independent random variables, where each term in the sum is closely related to the corresponding (t,x) -relevant experiments. By our careful choice of $\mathbb{P}[\hat{X}_i^{(t,x)} = 1]$, we can relate $\widehat{\deg}_{A^t}(x)$ to $\overline{\deg}_{A^t}(x)$ via the notion of *stochastic dominance* defined below.

► **Definition 19.** Let Y and Z be two random variables not necessarily defined on the same probability space. We say that Z stochastically dominates Y , written as $Y \preceq Z$, if for all $\lambda \in \mathbb{R}$, we have $\mathbb{P}[Y \geq \lambda] \leq \mathbb{P}[Z \geq \lambda]$.

Our second important step is the following lemma. The proof of Lemma 20 is omitted and will appear in our full version.

► **Lemma 20** (Key Step 2). $\overline{\deg}_{A^t}(x) \preceq \widehat{\deg}_{A^t}(x)$.

Lemma 20 reduces the problem to bounding $\widehat{\deg}_{A^t}(x)$, which is indeed relatively easy to bound because it is a sum of independent random variables. The last key step of our proof does exactly this:

► **Lemma 21** (Key Step 3). $\widehat{\deg}_{A^t}(x) \leq 2 \log(t) \cdot \text{target}^t(x) + O(\log|M|)$ with probability $1 - 1/|M|^{10}$.

We show the proof Lemma 21 in our full version. Here, we only mention one important point about the proof. The $\log(t)$ factor above follows from the fact the number of (t,e) -relevant experiment is always at most $\text{Rel}(t,e) \leq \log(t)$ for any time t and edge e . This property is so crucial and, actually, is what the proactive resampling technique is designed for.

Given three key steps above (Lemmas 18, 20, and 21), we can conclude the proof of Lemma 14.

Proof of Lemma 14. Recall that we ultimately want to show that, for every timestep t , the maintained assignment A^t has overhead $O(\log(T), \log|M|)$. In other words, for every $t \in T$ and every $x \in M$, we want to show that

$$\deg_{A^t}(x) \leq \text{target}^t(x) \cdot O(\log(T)) + O(\log|M|).$$

By Lemma 18, it suffices to show that

$$\overline{\deg}_{A^t}(x) \leq \text{target}^t(x) \cdot O(\log(T)) + O(\log|M|).$$

By Lemmas 20 and 21,

$$\begin{aligned} & \mathbb{P}[\overline{\deg}_{A^t}(x) \geq 2\log(t) \cdot \text{target}^t(x) + O(\log|M|)] \\ & \leq \mathbb{P}[\widehat{\deg}_{A^t}(x) \geq 2\log(t) \cdot \text{target}^t(x) + O(\log|M|)] && \text{(Lemma 20)} \\ & \leq 1/|M|^{10}. && \text{(Lemma 21)} \end{aligned}$$

Now we apply union bound to the probability above. There are $T \leq |M|$ timesteps and $|M|$ machines, hence the probability that a bad event happens is bounded by $\frac{T|M|}{|M|^{10}} = \frac{1}{|M|^8}$. Here, we conclude the proof of Lemma 14. \blacktriangleleft

7 Conclusion

In this paper, we study fully dynamic spanner algorithms against an adaptive adversary. Our algorithm in Theorem 1 maintains a spanner with near-optimal stretch-size trade-off using only $O(\log n)$ amortized recourse. This closes the current oblivious-vs-adaptive gap with respect to amortized recourse. Whether the gap can be closed for *worst-case recourse* is an interesting open problem.

The ultimate goal is to show algorithms against an adaptive adversary with polylogarithmic amortized update time or even worst-case. Via the multiplicative weight update framework [39, 41], such algorithms would imply $O(k)$ -approximate multi-commodity flow algorithm with $\tilde{O}(n^{2+1/k})$ time which would in turn improve the state-of-the-art. We made partial progress toward this goal by showing the first dynamic 3-spanner algorithms against an adaptive adversary with $\tilde{O}(\sqrt{n})$ update time in Theorem 3 and *simultaneously* with $\tilde{O}(1)$ amortized recourse in Theorem 2, improving upon the $O(n)$ amortized update time since the 15-year-old work by [5].

Generalizing our Theorem 3 to dynamic $(2k - 1)$ -spanners of size $\tilde{O}(n^{1+1/k})$, for any $k \geq 2$, is also a very interesting and challenging open question.

References

- 1 Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Apmf < amsp? gomory-hu tree for unweighted graphs in almost-quadratic time. *arXiv preprint*, 2021. [arXiv:2106.02981](https://arxiv.org/abs/2106.02981).
- 2 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *FOCS*, pages 335–344, 2016. [doi:10.1109/FOCS.2016.44](https://doi.org/10.1109/FOCS.2016.44).
- 3 Noga Alon, Omri Ben-Eliezer, Yuval Dagan, Shay Moran, Moni Naor, and Eylon Yogev. Adversarial laws of large numbers and optimal regret in online classification. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 447–455, 2021.
- 4 Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.
- 5 Giorgio Ausiello, Paolo Giulio Franciosa, and Giuseppe F. Italiano. Small stretch spanners on dynamic graphs. *J. Graph Algorithms Appl.*, 10(2):365–385, 2006. Announced at ESA’05. [doi:10.7155/jgaa.00133](https://doi.org/10.7155/jgaa.00133).

- 6 Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Dynamic balanced graph partitioning. *SIAM Journal on Discrete Mathematics*, 34(3):1791–1812, 2020.
- 7 Raef Bassily, Kobbi Nissim, Adam D. Smith, Thomas Steinke, Uri Stemmer, and Jonathan R. Ullman. Algorithmic stability for adaptive data analysis. *SIAM J. Comput.*, 50(3), 2021. doi:10.1137/16M1103646.
- 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. Announced at SODA’08. doi:10.1145/2344422.2344425.
- 9 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 382–405, 2019. doi:10.1109/FOCS.2019.00032.
- 10 Omri Ben-Eliezer, Rajesh Jayaram, David P Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*, pages 63–80, 2020.
- 11 Aaron Bernstein. Deterministic partially dynamic single source shortest paths in weighted graphs. In *ICALP*, volume 80, pages 44:1–44:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.44.
- 12 Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. *arXiv preprint*, 2020. arXiv:2004.08432.
- 13 Aaron Bernstein and Shiri Chechik. Deterministic decremental single source shortest paths: beyond the $o(mn)$ bound. In *STOC*, pages 389–397, 2016.
- 14 Aaron Bernstein and Shiri Chechik. Deterministic partially dynamic single source shortest paths for sparse graphs. In *SODA*, pages 453–469, 2017.
- 15 Aaron Bernstein and Shiri Chechik. Incremental topological sort and cycle detection in expected total time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 21–34, 2018. doi:10.1137/1.9781611975031.2.
- 16 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In *SODA*, pages 1899–1918, 2019.
- 17 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. *arXiv preprint*, 2021. arXiv:2101.07149.
- 18 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *STOC*, pages 365–376, 2019.
- 19 Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2830–2842. SIAM, 2021.
- 20 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *SODA*, 2015.
- 21 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *STOC*, pages 398–411, 2016.
- 22 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *SODA*, pages 470–489, 2017.
- 23 Sayan Bhattacharya and Peter Kiss. Deterministic rounding of dynamic fractional matchings. *arXiv preprint*, 2021. arXiv:2105.01615.
- 24 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.
- 25 Greg Bodwin and Sebastian Krinninger. Fully dynamic spanners with worst-case update time. In *ESA*, pages 17:1–17:18, 2016. doi:10.4230/LIPIcs.ESA.2016.17.

- 26 Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and l_1 -regression in nearly linear time for dense instances. In *STOC*, pages 859–869. ACM, 2021.
- 27 Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *FOCS*, pages 919–930. IEEE, 2020.
- 28 Vladimir Braverman, Avinatan Hassidim, Yossi Matias, Mariano Schain, Sandeep Silwal, and Samson Zhou. Adversarial robustness of streaming algorithms through importance sampling. *arXiv preprint*, 2021. [arXiv:2106.14952](https://arxiv.org/abs/2106.14952).
- 29 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. doi:10.1145/2933057.2933083.
- 30 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 370–381, 2019. doi:10.1109/FOCS.2019.00031.
- 31 Shiri Chechik and Tianyi Zhang. Dynamic low-stretch spanning trees in subpolynomial time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 463–475. SIAM, 2020.
- 32 Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1135–1146. IEEE, 2020.
- 33 Julia Chuzhoy. Decremental all-pairs shortest paths in deterministic near-linear time. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 626–639, 2021. doi:10.1145/3406325.3451025.
- 34 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. *CoRR*, abs/1910.08025, 2019.
- 35 Julia Chuzhoy and Sanjeev Khanna. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems. In *STOC*, pages 389–400, 2019.
- 36 Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2478–2496. SIAM, 2021. doi:10.1137/1.9781611976465.147.
- 37 Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. Preserving statistical validity in adaptive data analysis. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 117–126, 2015.
- 38 Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Trans. Algorithms*, 7(2):20:1–20:17, 2011. Announced at ICALP'07. doi:10.1145/1921659.1921666.
- 39 Lisa Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000. announced at FOCS'99. doi:10.1137/S0895480199355754.
- 40 Sebastian Forster and Gramoz Goranci. Dynamic low-stretch trees via dynamic low-diameter decompositions. In *STOC*, pages 377–388. ACM, 2019.
- 41 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007. doi:10.1137/S0097539704446232.

- 42 Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2212–2228. SIAM, 2021. doi:10.1137/1.9781611976465.132.
- 43 Ofer Grossman and Merav Parter. Improved deterministic distributed construction of spanners. In *DISC*, pages 24:1–24:16, 2017. doi:10.4230/LIPIcs.DISC.2017.24.
- 44 Anupam Gupta and Amit Kumar. Online steiner tree with deletions. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 455–467. SIAM, 2014.
- 45 Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: Matching, scheduling, and flows. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 468–479. SIAM, 2014.
- 46 Anupam Gupta and Roie Levin. Fully-dynamic submodular cover with bounded recourse. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1147–1157. IEEE, 2020.
- 47 Varun Gupta, Christopher Jung, Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi, and Chris Waites. Adaptive machine unlearning. *arXiv preprint*, 2021. arXiv:2106.04378.
- 48 Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Symposium on Theory of Computing*, 2020. arXiv:2001.10751.
- 49 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Decremental SSSP in weighted digraphs: Faster and against an adaptive adversary. In *SODA*, pages 2542–2561, 2020.
- 50 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Deterministic algorithms for decremental approximate shortest paths: Faster and simpler. In *SODA*, pages 2522–2541, 2020.
- 51 Moritz Hardt and Jonathan Ullman. Preventing false discovery in interactive data analysis is hard. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 454–463. IEEE, 2014.
- 52 Avinatan Hasidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. *Advances in Neural Information Processing Systems*, 33, 2020.
- 53 Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.
- 54 Jacob Holm and Eva Rotenberg. Fully-dynamic planarity testing in polylogarithmic time. In *STOC*, pages 167–180. ACM, 2020.
- 55 Jacob Holm and Eva Rotenberg. Worst-case polylog incremental spqr-trees: Embeddings, planarity, and triconnectivity. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2378–2397, 2020. doi:10.1137/1.9781611975994.146.
- 56 Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer. Separating adaptive streaming from oblivious streaming. *arXiv preprint*, 2021. arXiv:2101.10836.
- 57 Jason Li. Deterministic mincut in almost-linear time. In *STOC*, pages 384–395. ACM, 2021.
- 58 Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. *arXiv preprint*, 2021. arXiv:2106.02233.
- 59 Jason Li and Thatchaphol Saranurak. Deterministic weighted expander decomposition in almost-linear time. *arXiv preprint*, 2021. arXiv:2106.01567.
- 60 Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, Las vegas, and $O(n^{1/2-\epsilon})$ -time. In *STOC*, pages 1122–1129, 2017. doi:10.1145/3055399.3055447.
- 61 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *FOCS*, pages 950–961, 2017.

- 62 Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *SODA*, pages 2616–2635, 2019.
- 63 Thomas Steinke and Jonathan Ullman. Tight lower bounds for differentially private selection. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 552–563. IEEE, 2017.
- 64 David Wajc. Rounding dynamic matchings against an adaptive adversary. *Symposium on Theory of Computing*, 2020. [arXiv:1911.05545](https://arxiv.org/abs/1911.05545).
- 65 Rephael Wenger. Extremal graphs with no C_4 's, C_6 's, or C_{10} 's. *Journal of Combinatorial Theory, Series B*, 52(1):113–116, 1991.
- 66 David P Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. *arXiv preprint*, 2020. [arXiv:2011.07471](https://arxiv.org/abs/2011.07471).
- 67 Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *STOC*, pages 1130–1143, 2017. [doi:10.1145/3055399.3055415](https://doi.org/10.1145/3055399.3055415).
- 68 Tianyi Zhang. Faster cut-equivalent trees in simple graphs. *arXiv preprint*, 2021. [arXiv:2106.03305](https://arxiv.org/abs/2106.03305).

Online Spanners in Metric Spaces

Sujoy Bhore  



Indian Institute of Science Education and Research, Bhopal, India

Arnold Filtser  

Department of Computer Science, Bar-Ilan University, Ramat-Gan Israel

Hadi Khodabandeh  

Department of Computer Science, University of California, Irvine, CA, USA

Csaba D. Tóth  

California State University Northridge, Los Angeles, CA, USA

Tufts University, Medford, MA, USA

Abstract

Given a metric space $\mathcal{M} = (X, \delta)$, a weighted graph G over X is a metric t -spanner of \mathcal{M} if for every $u, v \in X$, $\delta(u, v) \leq \delta_G(u, v) \leq t \cdot \delta(u, v)$, where δ_G is the shortest path metric in G . In this paper, we construct spanners for finite sets in metric spaces in the online setting. Here, we are given a sequence of points (s_1, \dots, s_n) , where the points are presented one at a time (i.e., after i steps, we have seen $S_i = \{s_1, \dots, s_i\}$). The algorithm is allowed to add edges to the spanner when a new point arrives, however, it is not allowed to remove any edge from the spanner. The goal is to maintain a t -spanner G_i for S_i for all i , while minimizing the number of edges, and their total weight.

Under the L_2 -norm in \mathbb{R}^d for arbitrary constant $d \in \mathbb{N}$, we present an online $(1 + \varepsilon)$ -spanner algorithm with competitive ratio $O_d(\varepsilon^{-d} \log n)$, improving the previous bound of $O_d(\varepsilon^{-(d+1)} \log n)$. Moreover, the spanner maintained by the algorithm has $O_d(\varepsilon^{1-d} \log \varepsilon^{-1}) \cdot n$ edges, almost matching the (offline) optimal bound of $O_d(\varepsilon^{1-d}) \cdot n$. In the plane, a tighter analysis of the same algorithm provides an almost quadratic improvement of the competitive ratio to $O(\varepsilon^{-3/2} \log \varepsilon^{-1} \log n)$, by comparing the online spanner with an instance-optimal spanner directly, bypassing the comparison to an MST (i.e., lightness). As a counterpart, we design a sequence of points that yields a $\Omega_d(\varepsilon^{-d})$ lower bound for the competitive ratio for online $(1 + \varepsilon)$ -spanner algorithms in \mathbb{R}^d under the L_1 -norm.

Then we turn our attention to online spanners in general metrics. Note that, it is not possible to obtain a spanner with stretch less than 3 with a subquadratic number of edges, even in the offline setting, for general metrics. We analyze an online version of the celebrated greedy spanner algorithm, dubbed *ordered greedy*. With stretch factor $t = (2k - 1)(1 + \varepsilon)$ for $k \geq 2$ and $\varepsilon \in (0, 1)$, we show that it maintains a spanner with $O(\varepsilon^{-1} \log \varepsilon^{-1}) \cdot n^{1+\frac{1}{k}}$ edges and $O(\varepsilon^{-1} n^{\frac{1}{k}} \log^2 n)$ lightness for a sequence of n points in a metric space. We show that these bounds cannot be significantly improved, by introducing an instance that achieves an $\Omega(\frac{1}{k} \cdot n^{1/k})$ competitive ratio on both sparsity and lightness. Furthermore, we establish the trade-off among stretch, number of edges and lightness for points in ultrametrics, showing that one can maintain a $(2 + \varepsilon)$ -spanner for ultrametrics with $O(\varepsilon^{-1} \log \varepsilon^{-1}) \cdot n$ edges and $O(\varepsilon^{-2})$ lightness.

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 spanner, online algorithm, lightness, sparsity, minimum weight

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.18

Related Version *Full Version*: <https://arxiv.org/abs/2202.09991>

Funding *Csaba D. Tóth*: NSF DMS-1800734.



© Sujoy Bhore, Arnold Filtser, Hadi Khodabandeh, and Csaba D. Tóth; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 18; pp. 18:1–18:20

Leibniz International Proceedings in Informatics



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

1 Introduction

Let $\mathcal{M} = (P, \delta)$ be a finite metric space. Let $G = (P, E)$ be a graph on the points of P in \mathcal{M} , where the edges are weighted with the distances between their endpoints. The graph G is a t -spanner, for $t \geq 1$, if $\delta_G(u, v) \leq t \cdot \delta(u, v)$ for all $u, v \in P$, where $\delta_G(u, v)$ is the length of the shortest path between u and v in G , and $\delta(u, v)$ is the distance between u and v in \mathcal{M} .¹ The *stretch factor* t of G is the maximum distortion between the metrics δ and δ_G . Spanners were first introduced by Peleg and Schäffer [52], and since then they have turned out to be one of the fundamental graph structures with numerous applications in the area of distributed systems and communication, distributed queuing protocol, compact routing schemes, etc. [25, 43, 53, 54].

The study of Euclidean spanners, where $P \subset \mathbb{R}^d$ with L_2 -norm, was initiated by Chew [23]. Since then a large body of research has been devoted to Euclidean spanners due to its vast range of applications across domains, such as topology control in wireless networks, efficient regression in metric spaces, approximate distance oracles, data structures, and many more [34, 38, 57, 60]. Some of the results generalize to metric spaces with constant doubling dimensions [18] (the doubling dimension of \mathbb{R}^d under L_2 -norm is $\Theta(d)$).

Lightness and *sparsity* are two fundamental parameters for spanners. The lightness of a spanner $G = (P, E)$ is the ratio $w(G)/w(MST)$ between the total weight of G and the weight of a minimum spanning tree (MST) on P . The sparsity of G is the ratio $|E(G)|/|E(MST)| \approx |E(G)|/|P|$ between the number of edges of G and an MST. Since every spanner is connected and thus contain a spanning tree, the lightness and sparsity of a spanner G , resp., are trivial lower bounds for the ratio of $w(G)$ and $|E(G)|$ to the optimum weight and the number of edges.

Online Spanners. We are given a sequence of points (s_1, \dots, s_n) in a metric space, where the points are presented one-by-one, i.e., point s_i is revealed at step i , and $S_i = \{s_1, \dots, s_i\}$ for $i \in \{1, \dots, n\}$. The objective of an online algorithm is to maintain a t -spanner G_i for S_i for all i . The algorithm is allowed to *add* edges to the spanner when a new point arrives, however it is not allowed to *remove* any edge from the spanner. Moreover, the algorithm does not know the total number of points in advance.

The performance of an online algorithm ALG is measured by comparing it with the offline optimum OPT using the standard notion of competitive ratio [17, Ch. 1]. The *competitive ratio* of an algorithm ALG is defined as $\sup_{\sigma} \frac{ALG(\sigma)}{OPT(\sigma)}$, where the supremum is taken over all input sequences σ , $OPT(\sigma)$ is the minimum weight of a t -spanner for the (unordered) set of points in σ , and $ALG(\sigma)$ denotes the weight of the t -spanner produced by ALG for this input sequence. Note that, in order to measure the competitive ratio it is important that σ is a finite sequence of points.

The online spanner problem is motivated by natural application domains. For example, consider a developing area with limited resources, where new settlements are created successively. As the community grows, new roads are built, and there is no reason to remove existing roads. Alternatively, online spanners are also motivated by distributed mobile computing, where new subscribers successively join a network. Maintaining a cost-effective network is equivalent to minimum-weight online spanner problem.

¹ Often in the literature, the input metric is the shortest path metric of a graph $G = (V, E, w)$, and a spanner is required to be a subgraph of the input graph (see e.g. [4]). Here we study metric spanners where there is no such requirement.

In the online minimum spanning tree problem, points of a finite metric space arrive one-by-one, and we need to connect each new point to a previous point to maintain a spanning tree. Imase and Waxman [44] proved $\Theta(\log n)$ -competitiveness, which is the best possible bound. Later, Alon and Azar [2] studied this problem 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 analyze the impact of auxiliary points (Steiner points) on a geometric network problem in the online setting. Several algorithms were proposed over the years for the online minimum Steiner tree and Steiner forest problems, on graphs in both weighted and unweighted settings; see [1, 5, 10, 40, 50]. However, these algorithms do not provide any guarantees on the stretch factor. This leads to the following open problem.

► **Problem.** *Determine the best possible bounds for the competitive ratios for the weight and the number of edges of online t -spanners, for $t \geq 1$.*

Previously, Gupta et al. [39, Theorem 1.5] constructed online spanners for terminal pairs in the same model we consider here. The analysis of [39] implicitly implies that, given a sequence of n points in an online fashion in a general metric space, one can maintain a $O(\log n)$ -spanner with $O(n)$ edges and $O(\log n)$ lightness, as pointed out by one of the authors [59]. Recent work on online *directed* spanners [36] is not comparable to our results.

In the geometric setting, $(1 + \varepsilon)$ -spanners are possible in any constant dimension $d \in \mathbb{N}$. Tight worst-case bounds $\Theta_d(\varepsilon^{-d})$ and $\Theta_d(\varepsilon^{1-d})$ on the lightness and sparsity of offline $(1 + \varepsilon)$ -spanners have recently been established by Le and Solomon [47]. Online Euclidean spanners in \mathbb{R}^d have been introduced by Bhore and Tóth [14]. In the real line (1D), they have established a tight bound of $O((\varepsilon^{-1} / \log \varepsilon^{-1}) \log n)$ for the competitive ratio of any online $(1 + \varepsilon)$ -spanner algorithm for n points. In dimensions $d \geq 2$, the dynamic algorithm DEFSPANNER of Gao et al. [33] maintains a $(1 + \varepsilon)$ -spanner with $O_d(\varepsilon^{-(d+1)}n)$ edges and $O_d(\varepsilon^{-(d+1)} \log n)$ lightness, and works under the online model (as it never deletes edges when new points arrive). However, no lower bound better than the 1-dimensional $\Omega((\varepsilon^{-1} / \log \varepsilon^{-1}) \log n)$ is currently known in higher dimensions.

1.1 Our Contribution

See Table 1 for an overview of our results.

Upper Bounds for Points in \mathbb{R}^d . Under the L_2 -norm in \mathbb{R}^d , for arbitrary constant $d \in \mathbb{N}$, we present an online algorithm for $(1 + \varepsilon)$ -spanner with lightness $O_d(\varepsilon^{-d} \log n)$ and sparsity $O(\varepsilon^{1-d} \log \varepsilon^{-1})$ (Theorem 2 in Section 2.1). This improves upon the previous lightness bound of $O_d(\varepsilon^{-(d+1)} \log n)$ by Gao et al. [33, Lemma 3.8]. In the plane, we give a tighter analysis of the same algorithm and achieve an almost quadratic improvement of the *competitive ratio* to $O(\varepsilon^{-3/2} \log \varepsilon^{-1} \log n)$ (Theorem 6 in Section 2.2). Recall that in the offline setting, $\Theta(\varepsilon^{-2})$ is a tight worst-case bound for the lightness of a $(1 + \varepsilon)$ -spanner in the plane [47]. We obtain a better dependence on ε by comparing the online spanner with an instance-optimal spanner directly, bypassing the comparison to an MST (i.e., lightness). The logarithmic dependence on n cannot be eliminated in the online setting, based on the lower bound in \mathbb{R}^1 [14].

Lower Bounds for Points in \mathbb{R}^d . As a counterpart, we design a sequence of points that yields a $\Omega_d(\varepsilon^{-d})$ lower bound for the competitive ratio for online $(1 + \varepsilon)$ -spanner algorithms in \mathbb{R}^d under the L_1 -norm (Theorem 13 in Section 3). This improves the previous bound of $\Omega(\varepsilon^{-2} / \log \varepsilon^{-1})$ in \mathbb{R}^2 under the L_1 -norm [14]. It remains open whether a similar lower bound holds in \mathbb{R}^d under the L_2 -norm; the current best lower bound is $\Omega((\varepsilon^{-1} / \log \varepsilon^{-1}) \log n)$, established in [14], holds already for the real line ($d = 1$).

■ **Table 1** Overview of online spanners algorithms. In the last three rows, we compare the spanner weight directly with the optimum weight (rather than the MST) to bound the competitive ratio.

Family	Stretch	# of edges	Lightness	Reference
General metrics	$(2k - 1)(1 + \varepsilon)$	$O(\varepsilon^{-1} \log \varepsilon^{-1}) n^{1+\frac{1}{k}}$	$O(\varepsilon^{-1} n^{\frac{1}{k}} \log^2 n)$	Theorem 14
	$O(\log n)$	$O(n)$	$O(\log n)$	[39, 59]
α -HST	$2 \frac{\alpha}{\alpha-1}$	$n - 1$	1	Full paper
Ultrametric	$O(\varepsilon^{-1})$	$n - 1$	$1 + \varepsilon$	Full paper
	$2 + \varepsilon$	$O(\varepsilon^{-1} \log \varepsilon^{-1}) n$	$O(\varepsilon^{-2})$	Full paper
Doubling d -space	$1 + \varepsilon$	$\varepsilon^{-O(d)} n$	$\varepsilon^{-O(d)} \log n$	[33]
Euclidean d -space	$1 + \varepsilon$	$O_d(\varepsilon^{-d}) n$	$O_d(\varepsilon^{-(d+1)} \log n)$	[33]
	$1 + \varepsilon$	$O_d(\varepsilon^{1-d}) n$	$\Omega(\varepsilon^{-1} n)$	[56]
	$1 + \varepsilon$	$O_d(\varepsilon^{1-d} \log \varepsilon^{-1}) n$	$O_d(\varepsilon^{-d} \log n)$	Theorem 2
Real line	$1 + \varepsilon$	$O(n)$	$\tilde{O}(\varepsilon^{-1} \log n)$	[14]
Family	Stretch	# of edges	Comp. Ratio	Reference
General metrics	$2k - 1$	-	$\Omega(\frac{1}{k} \cdot n^{\frac{1}{k}})$	Theorem 19
Euclidean plane	$1 + \varepsilon$	$O(\varepsilon^{-1} \log \varepsilon^{-1}) n$	$\tilde{O}(\varepsilon^{-3/2} \log n)$	Theorem 6
\mathbb{R}^d with L_1 -norm	$1 + \varepsilon$	-	$\Omega(\varepsilon^{-d})$	Theorem 13

Points in General Metrics. In Section 4, we study online spanners in general metrics. Note that it is not possible to obtain a spanner with stretch less than 3 with a subquadratic number of edges, even in the offline settings, for general metrics. We analyze an online version of the celebrated greedy spanner algorithm, dubbed *ordered greedy*. With stretch factor $t = (2k - 1)(1 + \varepsilon)$ for $k \geq 2$ and $\varepsilon \in (0, 1)$, we show that it maintains a spanner with $O(\varepsilon^{-1} \log \varepsilon^{-1}) \cdot n^{1+\frac{1}{k}}$ edges and $O(\varepsilon^{-1} n^{\frac{1}{k}} \log^2 n)$ lightness for a sequence of n points in a metric space (Theorem 14). We show (in Theorem 19) that these bounds cannot be significantly improved, by introducing an instance where every online algorithm will have $\Omega(\frac{1}{k} \cdot n^{1/k})$ competitive ratio on both sparsity and lightness. Next, we establish the trade-off among stretch, number of edges and lightness for points in ultrametrics. Specifically, we show (in the full paper) that it is possible to maintain a $(2 + \varepsilon)$ -spanner with $O(\varepsilon^{-1} \log \varepsilon^{-1}) \cdot n$ edges and $O(\varepsilon^{-2})$ lightness in ultrametrics. Note that as the uniform metric (shortest path on a clique) is an ultrametric, any subquadratic spanner must have stretch at least 2.

1.2 Related Work

1.2.1 Dynamic & Streaming Algorithms for Graph Spanners

A t -spanner in a graph $G = (V, E)$ is a subgraph $H = (V, E')$ such that $\delta_H(u, v) \leq t \cdot \delta_G(u, v)$ for all pairs of vertices $u, v \in V$. That is, the stretch t is the maximum distortion between the graph distances δ_G and δ_H . Importantly, when G changes (under edge/vertex insertions or deletions), the underlying metric δ_G changes, as well. The distance $\delta_G(u, v)$ may dramatically decrease upon the insertion of the edge uv . In contrast, our model assumes that the distances in the underlying metric space $\mathcal{M} = (P, \delta)$ remain fixed, but the algorithm can only see the distances between the points that have been presented. For this reason, our results are not directly comparable to models where the underlying graph changes dynamically.

For *unweighted* graphs with n vertices, the current best fully dynamic and single-pass streaming algorithms can maintain spanners that achieve almost the same stretch-sparsity trade-off available for the static case: $2k - 1$ stretch and $O(n^{1+\frac{1}{k}})$ edges, for $k \geq 1$, which is

attained by the greedy algorithm [4], and conjectured to be optimal due to the Erdős girth conjecture [28]. 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. See [7, 9, 11, 16] for some excellent work on dynamic spanners. In the streaming model the input is a sequence (or stream) of edges representing the edge set E of the graph G . A (single-pass) streaming algorithm decides, for each newly arriving edge, whether to include it in the spanner. The graph G is too large to fit in memory, and the objective is to optimize work space and update time [6, 8, 26, 29, 30, 49].

1.2.2 Incremental Algorithms for Geometric Spanners

We briefly review three previously known incremental $(1 + \varepsilon)$ -spanner algorithms in Euclidean d -space from the perspective of competitive analysis.

Deformable Spanners. Gao et al. [33] designed a dynamic DEFSPANNER algorithm that maintains a $(1 + \varepsilon)$ -spanner for a dynamic set S in the Euclidean d -space. For point insertions, it only adds new edges, so it is an online algorithm, as well. It maintains a $(1 + \varepsilon)$ -spanner with $O_d(\varepsilon^{-d}) \cdot n$ edges and $O_d(\varepsilon^{-(d+1)} \log n)$ lightness. Since the $\|\text{MST}(S)\|$ is a lower bound for the optimal spanner weight, its competitive ratio is also $O_d(\varepsilon^{-(d+1)} \log n)$. The key ingredient of DEFSPANNER is *hierarchical nets* [42, 46, 55], a form of hierarchical clustering, which can be maintained dynamically. Hierarchical nets naturally generalize to doubling spaces, and so DEFSPANNER also maintains a $(1 + \varepsilon)$ -spanner with $\varepsilon^{-O(d)} \cdot n$ edges and $\varepsilon^{-O(d)} \cdot \log n$ lightness for doubling dimension d [35, 55].

Well-Separated Pair Decomposition (WSPD). Well-separated pair decomposition was introduced by Callahan and Kosaraju [21] (see also [37, 41, 51, 58]). For a set S in a metric space, a WSPD is a collection of unordered pairs $W = \{\{A_i, B_i\} : i \in I\}$ such that (1) $A_i, B_i \subset S$ for all $i \in I$; (2) $\min\{\|ab\| : a \in A_i, b \in B_i\} \leq \varrho \cdot \max\{\text{diam}(A_i), \text{diam}(B_i)\}$ for all $i \in I$, where ϱ is the *separation ratio*; (3) for each point pair $\{a, b\} \subset S$ there exists a pair $\{A_i, B_i\}$ such that A_i and B_i each contain one of a and b . Given a WSPD with separation ratio $\varrho > 4$, any graph that contains at least one edge between A_i and B_i , for all $i \in I$, is a spanner with stretch $t = 1 + 8/(\varrho - 4)$. Setting $\varrho \geq 12\varepsilon^{-1}$ for $0 < \varepsilon < 1$, we obtain $t \leq 1 + \varepsilon$.

Hierarchical clustering provides a WSPD [41, Ch. 3]. Perhaps the simplest hierarchical subdivisions in \mathbb{R}^d are quadtrees. Let \mathcal{T} be a quadtree for a finite set $S \subset \mathbb{R}^d$. The root of \mathcal{T} is an axis-aligned cube of side length a_0 , which contains S ; it is recursively subdivided into 2^d congruent cubes until each leaf cube contains at most one point in S . For all pairs of cubes $\{Q_1, Q_2\}$ at level ℓ of \mathcal{T} , create a pair $\{A_i, B_i\}$ with $A_i = Q_1 \cap S$ and $B_i = Q_2 \cap S$ whenever $D_\ell \leq \text{dist}(Q_1, Q_2) < 2D_\ell$ for $D_\ell = \varrho \cdot \text{diam}(Q_1) = 12\varepsilon^{-1} \cdot \sqrt{d} \cdot a_0 / 2^\ell$; and repeat for all levels $\ell \geq 0$. Properties (1)–(3) of a WSPD are easily verified [41, Ch. 3]. The resulting $(1 + \varepsilon)$ -spanner has $O_d(\varepsilon^{-d}) \cdot n$ edges [41, 42] and lightness $O_d(\varepsilon^{-(d+1)} \log n)$ [14].

For point insertions in \mathbb{R}^d , a dynamic quadtree only adds nodes, which in turn creates new pairs in the WSPD, and new edges in the spanner. This is an online algorithm with the same guarantees as DEFSPANNER [14, 42] (see also [32] for an efficient implementation).

Ordered Yao-Graphs and Θ -Graphs. Among the first constructions for (offline) sparse $(1 + \varepsilon)$ -spanners in the Euclidean d -space were Yao- and Θ -graphs [24, 45, 56]. Incremental versions of Yao-graphs and Θ -graphs were introduced by Bose et al. [20]. Let $S = \{s_1, \dots, s_n\}$ be an ordered set of points in \mathbb{R}^2 . For each $s_i \in S$, partition the plane into k cones with apex s_i and aperture $2\pi/k$. The *ordered Yao-graph* $Y_k(S)$ contains an edge between s_i and

a closest *previous* point in $\{s_j : j < i\}$ in each cone. The graph $\Theta_k(S)$ is defined similarly, but in each cone the distance to the apex is measured by the orthogonal projection to a ray within the cone. Bose et al. [20] showed that the ordered Yao- and Θ -graphs have spanning ratio at most $1/(1 - 2\sin(\pi/k))$ for $k > 8$; tighter bounds were later obtained in [19]. In particular, the ordered Yao- and Θ -graphs are $(1 + \varepsilon)$ -spanners for $k \geq \Omega(\varepsilon^{-1})$.

The construction generalizes to \mathbb{R}^d for all $d \in \mathbb{N}$ [56]. For an angle $\alpha \in (0, \pi)$, let $A \subset \mathbb{S}^{d-1}$ be a maximal set of points in the $(d-1)$ -sphere such that $\min_{a,b \in A} \text{dist}(a,b) \leq \alpha$ (in radians). A standard volume argument shows that $|A| \leq O_d(\alpha^{1-d})$. For each $a_i \in A$, create a cone C_i with apex at the origin o , aperture α , and symmetry axis oa_i . Note that $\mathbb{R}^d \subseteq \bigcup_i C_i$. Given a finite set $P \subset \mathbb{R}^d$, we translate each cone C_i to a cone $C_i(p)$ with apex $p \in P$. For every cone $C_i(p)$, the Yao-graph contains an edge between p and a closest point in $P \cap C_i(p)$. For every $\varepsilon > 0$ and $d \in \mathbb{N}$, there exists an angle $\alpha = \alpha(d, \varepsilon) = \Theta_d(\varepsilon)$ for which the Yao-graph is a $(1 + \varepsilon)$ -spanner for every finite set $P \subset \mathbb{R}^d$.

Ordered Yao- and Θ -graphs give online algorithms for maintaining a $(1 + \varepsilon)$ -spanner for a sequence of points in \mathbb{R}^d . The sparsity of these spanners is bounded by the number of cones per vertex, $O_d(\varepsilon^{1-d})$, which matches the (offline) lower bound of $\Omega_d(\varepsilon^{1-d})$ [47]. However, their weight may be significantly higher than optimal: For n equally spaced points in a unit circle, in any order, Yao- and Θ -graphs yield $(1 + \varepsilon)$ -spanners of weight $\Omega(\varepsilon^{-1}) \cdot n$, hence lightness $\Omega(\varepsilon^{-1}) \cdot n$, while the optimum weight is $O(\varepsilon^{-2})$ [47].

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 the input sequence of points, but are present in the metric space. An online algorithm is allowed *add* Steiner points, however, the spanner must achieve the given stretch factor only for the input point pairs. It has been observed through a series of work in recent years, that Steiner points allow for substantial improvements over the bounds on the sparsity and lightness of Euclidean spanners in the offline settings. Highly nontrivial insights are required to argue the bounds for Steiner spanners, and often they tend to be even more intricate than their non-Steiner counterpart; see [12, 13, 47, 48]. Bhore and Tóth [14] showed that if an algorithm can use Steiner points, then the competitive ratio for weight improves to $O(\varepsilon^{(1-d)/2} \log n)$ in the Euclidean d -space.

2 Upper Bounds in Euclidean Spaces

We present an online algorithm for a sequence of n points in the Euclidean d -space (Section 2.1). It combines features from several previous approaches, and maintains a $(1 + \varepsilon)$ -spanner of lightness $O_d(\varepsilon^{-d} \log n)$ and sparsity $O_d(\varepsilon^{1-d} \log \varepsilon^{-1})$ for $d \geq 1$. Lightness is an upper bound for the competitive ratio for weight; the sparsity almost matching the optimal bound $O_d(\varepsilon^{1-d})$ attained by ordered Yao-graphs. In the plane ($d = 2$), we show that the same algorithm achieves competitive ratio $O(\varepsilon^{-3/2} \log \varepsilon^{-1} \log n)$ using a tighter analysis: A charging scheme that charges the weight of the online spanner to a minimum weight spanner (Section 2.2).

2.1 An Improvement in All Dimensions

We combine features from two incremental algorithms for geometric spanners, and obtain an online $(1 + \varepsilon)$ -spanner algorithm for a sequence of n points in \mathbb{R}^d . We maintain a dynamic quadtree for hierarchical clustering, and use a modified ordered Yao-graph in each level of the hierarchy. In particular, we limit the weight of the edges in the Yao-graph in each level of the hierarchy (thereby avoiding heavy edges). We start with an easy observation.

► **Lemma 1.** *Let $G = (S, E)$ be a t -spanner and let $w > 0$. Let $G' = (S, E')$, where $E' = \{e \in E : \|e\| \leq w\}$ is the set of edges of weight at most w . Then for every $a, b \in S$ with $\|ab\| < w/t$, graph G' contains an ab -path of weights at most $t\|ab\|$.*

Proof. Since G is a t -spanner, it contains an ab -path P_{ab} of weight at most $t\|ab\| \leq w$. By the triangle inequality, every edge in this path has weight at most w , hence present in G' . Consequently G' contains P_{ab} . ◀

The input is a sequence of points (s_1, s_2, \dots) in \mathbb{R}^d , $d \geq 1$. 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 $a_\ell = a_0 2^{-\ell}$. A cube $Q \in \mathcal{T}_n$ is *nonempty* if $Q \cap S_n \neq \emptyset$. For every nonempty cube Q , we maintain a representative $s(Q) \in Q \cap S_n$, selected at the time when Q becomes nonempty. At each level ℓ , let P_ℓ be the sequence of representatives, in the order in which they are created.

For each level ℓ , we maintain a modified ordered Yao-graph $G_\ell = (P_\ell, E_\ell)$ as follows. When a new point p is inserted into P_ℓ , cover \mathbb{R}^d with $\Theta_d(\varepsilon^{1-d})$ cones of aperture $\alpha(d, \varepsilon)$ as in the construction of Yao-graphs. In each cone C_i , find a point $q_i \in C_i \cap P_\ell$ closest to p ; and add pq_i to E_ℓ if $\|pq_i\| < 24a_\ell\sqrt{d} \cdot \varepsilon^{-1}$. The algorithm maintains the spanner $G = \bigcup_{\ell \geq 0} G_\ell$.

► **Theorem 2.** *Let $d \geq 1$ and $\varepsilon \in (0, 1)$. The online algorithm ALG_1 maintains, for a sequence of n points in Euclidean d -space, an $(1 + O(\varepsilon))$ -spanner with weight $O_d(\varepsilon^{-d} \log n) \cdot \|MST\|$ and $O_d(\varepsilon^{1-d} \log \varepsilon^{-1}) \cdot n$ edges.*

Note that Theorem 2 implies that the competitive ratio of this algorithm is also $O_d(\varepsilon^{-d} \log n)$.

Proof.

Stretch Analysis. We give a bound on the stretch factor in two steps: First, we define an auxiliary graph $H = (S, E')$ which is a $(1 + \varepsilon)$ -spanner for S by the analysis of WSPDs. Then we show that G contains an ab -path of weight at most $(1 + \varepsilon)\|ab\|$ for each edge of H . Overall, the stretch of G is at most $(1 + \varepsilon)^2 = (1 + O(\varepsilon))$ for all $a, b \in S$.

First Layer: WSPD. For each level $\ell \geq 0$, let $H_\ell = (P_\ell, E'_\ell)$ be the graph that contains an edge between two representatives $a, b \in P_\ell$ whenever $\|ab\| \leq 12a_\ell\sqrt{d} \cdot \varepsilon^{-1}$. Let $H = \bigcup_{\ell \geq 0} H_\ell$. The auxiliary graph H_ℓ contains an edge between the representatives of any such pair of cubes at level ℓ . As noted Section 1.2.2, $H = \bigcup_{\ell \geq 0} H_\ell$ is a $(1 + \varepsilon)$ -spanner (cf. [41, 42]).

Second Layer: Near-Sighted Yao-graphs. As H is a $(1 + \varepsilon)$ -spanner, for every $a, b \in S_n$, it contains an ab -path of weight at most $(1 + \varepsilon)\|ab\|$. Consider such a path $P_{ab} = (a = p_0, \dots, p_m = b)$. Each edge $p_{i-1}p_i$ is in H_ℓ for some $\ell \geq 0$. By construction, every edge in H_ℓ has weight at most $12a_\ell\sqrt{d} \cdot \varepsilon^{-1}$. For every level ℓ , the ordered Yao-graph $Y(P_\ell)$ with angle $\alpha(d, \varepsilon)$ is a $(1 + \varepsilon)$ -spanner. The graph $G_\ell = (P_\ell, E_\ell)$ constructed by ALG_1 at level ℓ is a subgraph of $Y(P_\ell)$. By Lemma 1, for every $p, q \in P_\ell$ with $\|pq\| \leq 12a_\ell\sqrt{d} \cdot \varepsilon^{-1}$, graph G_ℓ contains a pq -path of weight at most $(1 + \varepsilon)\|pq\|$.

Overall, H contains an ab -path $P_{ab} = (p_0, \dots, p_m)$ of weight at most $(1 + \varepsilon)\|ab\|$. For each edge $p_{i-1}p_i$ of P_{ab} , graph G contains a $p_{i-1}p_i$ -path of weight $(1 + \varepsilon)\|p_{i-1}p_i\|$. The concatenation of these paths is an ab -path of weight $(1 + \varepsilon)^2\|ab\| \leq (1 + O(\varepsilon))\|ab\|$.

Weight Analysis. We may assume w.l.o.g. that the root of the quadtree \mathcal{T}_n is the unit cube $[0, 1]^d \subset \mathbb{R}^d$, which has diameter \sqrt{d} . This implies $\text{diam}(S_n) \leq \sqrt{d} = O_d(1)$. Assume further that $n > 1$, and $\frac{1}{4} \leq \text{diam}(S_n) \leq \|MST(S_n)\|$.

Every edge in E_ℓ at level ℓ has weight $O_d(\varepsilon^{-1} 2^{-\ell})$. In particular, every edge at level $\ell \geq 2 \log n$ has weight $O_d(\varepsilon^{-1}/n^2)$; and the total weight of these edges is $O_d(\varepsilon^{-1}) \leq O_d(\varepsilon^{-1} \|MST(S_n)\|)$.

It remains to bound the weight of the edges on levels $\ell = 1, \dots, \lfloor 2 \log n \rfloor$. At level ℓ of the quadtree \mathcal{T}_n , there are at most $2^{d\ell}$ nodes, hence $|P_\ell| \leq 2^{d\ell}$. If $|P_\ell| < 3^d$, then G_ℓ has at most $O(3^{2d}) = O_d(1)$ edges, each of weight at most $\text{diam}(P_\ell) \leq \text{diam}(S_n) \leq \|MST(S_n)\|$, and so $\|E_\ell\| \leq O_d(\|MST(S_n)\|)$. Assume now that $|G_\ell| \geq 3^d$. By the definition of ordered Yao-graphs, each vertex inserted into P_ℓ adds $\Theta(\varepsilon^{1-d})$ new edges, each of weight $O(\varepsilon^{-1} 2^{-\ell})$. The total weight of the edges in G_ℓ is at most

$$\|E_\ell\| \leq |P_\ell| \cdot \varepsilon^{1-d} \cdot \max_{e \in E_\ell} \|e\| \leq O_d(|P_\ell| \varepsilon^{-d} 2^{-\ell}). \quad (1)$$

We next derive a lower bound for $\|MST(S_n)\|$ in terms of $|P_\ell|$, when $|P_\ell| > 1$ and $\ell > 2$, using a standard volume argument. Define a graph on the vertex set P_ℓ such that two nodes $p, q \in P_\ell$ are adjacent iff p and q lie in neighboring quadtree cells of level ℓ . Since every quadtree cell has $3^d - 1$ neighbors, this graph is $(3^d - 1)$ -degenerate, and contains an independent set I_ℓ of size at least $(3^d - 1)^{-1} |P_\ell| = \Omega_d(|P_\ell|)$. The distance between any two disjoint quadtree cells at level ℓ is at least $2^{-\ell}$. Consequently, the open balls of radius $2^{-(\ell+1)}$ centered at the points in I_ℓ are pairwise disjoint. None of the balls contains S_n for $\ell > 2$, as the diameter of each of ball is $2^{-\ell}$ while $\text{diam}(S_n) \geq \frac{1}{4}$. For all $\ell > 2$, $MST(S_n)$ contains the center of each ball and a point in its exterior; hence the intersection of $MST(S_n)$ and each ball contains a path from the center to a boundary point, which has weight at least $2^{-(\ell+1)}$. Summation over $|I_\ell|$ disjoint balls yields

$$\|MST(S_n)\| \geq |I_\ell| \cdot 2^{-(\ell+1)} \geq \Omega_d(|P_\ell| 2^{-\ell}). \quad (2)$$

Comparing inequalities (1) and (2), we obtain $\|E_\ell\| \leq O_d(\varepsilon^{-d}) \cdot \|MST(S_n)\|$. Summation over all levels $\ell \in \mathbb{N}$ yields $\|E\| \leq O_d(\varepsilon^{-d} \log n) \cdot \|MST(S_n)\|$, as claimed.

Sparsity Analysis. In the full paper, we show that G has $O(\varepsilon^{1-d} \log \varepsilon^{-1}) \cdot n$ edges. ◀

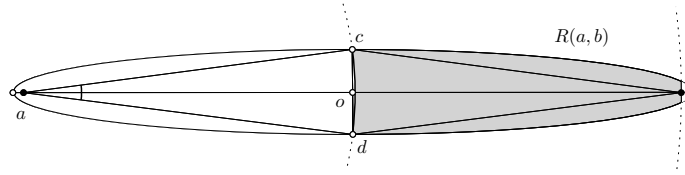
2.2 Further Improvements in the Plane

We presents a tighter analysis of algorithm ALG_1 for $d = 2$ that compares the spanner weight to the offline optimum weight, and bypasses the comparison with the MST (i.e., lightness).

Minimum-Weight Euclidean $(1 + \varepsilon)$ -Spanner. For any $a, b \in \mathbb{R}^d$, an ab -path P_{ab} of Euclidean weight at most $(1 + \varepsilon)\|ab\|$ lies in the ellipsoid \mathcal{E}_{ab} with foci a and b and great axes of weight $(1 + \varepsilon)\|ab\|$; see Figure 1. A key observation is that the minor axis of \mathcal{E}_{ab} is $((1 + \varepsilon)^2 - 1^2)^{1/2} \|ab\| \approx \sqrt{2\varepsilon} \|ab\|$. Furthermore, Bhore and Tóth [13] recently observed that the directions of “most” edges of the path P_{ab} are “close” to the direction of ab . Specifically, if we denote by $E(\alpha)$ the set of edges e in P_{ab} with $\angle(ab, e) \leq \alpha$, then the following holds.

► **Lemma 3** (Bhore and Tóth [13]). *Let $a, b \in \mathbb{R}^d$ and let P_{ab} be an ab -path of weight $\|P_{ab}\| \leq (1 + \varepsilon)\|ab\|$. Then for every $i \in \{1, \dots, \lfloor 1/\sqrt{\varepsilon} \rfloor\}$, we have $\|E(i \cdot \sqrt{\varepsilon})\| \geq (1 - 2/i^2) \|ab\|$.*

Let $R(a, b) = \mathcal{E}_{ab} \cap \mathcal{N}(a, b)$, where $\mathcal{N}(a, b)$ is the annulus bounded by two concentric spheres centered at a , of radii $\frac{1+\varepsilon}{2} \|ab\|$ and $\|ab\|$; see Figure 1 for an example.



■ **Figure 1** Any ab -path of weight at most $(1 + \varepsilon)\|ab\|$ lies in the ellipse \mathcal{E}_{ab} with foci a and b . The shaded region $R(a, b)$ is the part of the ellipse \mathcal{E}_{ab} between two concentric circles centered at a .

► **Lemma 4.** *If $0 < \varepsilon < \frac{1}{9}$, then every ab -path P_{ab} of weight at most $\|P_{ab}\| \leq (1 + \varepsilon)\|ab\|$ contains interior-disjoint line segments $s \subset R(a, b)$ of total weight at least $\frac{1}{9}\|ab\|$ such that $\angle(\vec{ab}, s) \leq 3 \cdot \sqrt{\varepsilon}$.*

Proof. Since the distance between the two concentric circles is $\frac{1-\varepsilon}{2}\|ab\|$, every ab -path contains a subpath of weight at least $\frac{1-\varepsilon}{2}\|ab\|$ in the annulus $\mathcal{N}(a, b)$.

Let P_{ab} be an ab -path of weight at most $(1 + \varepsilon)\|ab\|$. As noted above $P_{ab} \subset \mathcal{E}_{ab}$. Hence, $\|P_{ab} \cap \mathcal{N}(a, b)\| = \|P_{ab} \cap R(a, b)\| \geq \frac{1-\varepsilon}{2}\|ab\|$ in $R(a, b)$; and so $\|P_{ab} \setminus R(a, b)\| = \|P_{ab}\| - \|P_{ab} \cap R(a, b)\| \leq \frac{1+3\varepsilon}{2}\|ab\|$.

Applying Lemma 3 with $i = 3$, the total weight of the edges e of P_{ab} with $\text{dir}(ab, e) \leq 3 \cdot \sqrt{\varepsilon}$ is at least $\frac{7}{9}\|ab\|$. The parts of these edges lying outside of $R(a, b)$ have weight at most $\|P_{ab} \setminus R(a, b)\| \leq \frac{1+3\varepsilon}{2}\|ab\|$. Consequently, the remaining part of these edges are in $R(a, b)$, and their weight is at least $(\frac{7}{9} - \frac{1+3\varepsilon}{2})\|ab\| = \frac{5-27\varepsilon}{18}\|ab\| > \frac{1}{9}\|ab\|$ if $\varepsilon < \frac{1}{9}$, as claimed ◀

We also need an observation from elementary geometry; see Figure 1.

► **Lemma 5.** *For $a, b \in \mathbb{R}^d$, let cd be the minor axis of the ellipsoid \mathcal{E}_{ab} . Then $\angle cad \leq \sqrt{8\varepsilon}$.*

Proof. We may assume w.l.o.g. that $\|ab\| = 1$. Let o be the center of the ellipsoid \mathcal{E}_{ab} . Then $\sec \angle cao = (\cos \angle cao)^{-1} = \frac{\|ac\|}{\|ao\|} = 1 + \varepsilon$. The Taylor estimate $\sec(x) = 1 + \frac{1}{2}x^2 + \frac{5}{24}x^4 + \dots \geq 1 + \frac{1}{2}x^2$ for $0 < x < 1$ yields $\angle cao \leq \sqrt{2\varepsilon}$. Consequently, $\angle cad = 2\angle cao \geq \sqrt{8\varepsilon}$. ◀

► **Theorem 6.** *Let $d = 2$ and $\varepsilon \in (0, 1)$. The online algorithm ALG_1 maintains, for a sequence of n points in Euclidean plane, an $(1 + \varepsilon)$ -spanner of weight $O(\varepsilon^{-3/2} \log \varepsilon^{-1} \log n) \cdot \text{OPT}$, where OPT denotes the minimum weight of an $(1 + \varepsilon)$ -spanner for the same point set.*

Proof. Theorem 2 has established that algorithm ALG_1 maintains a $(1 + \varepsilon)$ -spanner. The tighter competitive analysis uses Lemmas 4 and 5.

Competitive Analysis. Assume w.l.o.g. that $\text{diam}(S_n) = \Theta(1)$, hence the side length of every quadtree square at level ℓ is $\Theta(2^{-\ell})$. For a set $S_n = \{s_1, \dots, s_n\} \subset \mathbb{R}^2$, let $G^* = (S_n, E^*)$ be a $(1 + \varepsilon)$ -spanner of minimum weight, and let $\text{OPT} = \|G^*\|$. Let $G = (S_n, E)$ be the spanner returned by the online algorithm ALG_1 . Recall that $G = \bigcup_{\ell \geq 0} G_\ell$, where the total weight of all edges at levels $\ell > 2 \log n$ is less than $\text{diam}(S_n)$, so it is enough to consider $\ell = 0, \dots, \lceil 2 \log n \rceil$.

▷ **Claim 7.** $\|G_\ell\| \leq O(\varepsilon^{-3/2} \log \varepsilon^{-1}) \cdot \text{OPT}$ for all $\ell \geq 0$.

Claim 7 immediately implies $\|G\| \leq O(\varepsilon^{-3/2} \log \varepsilon^{-1} \log n) \cdot \text{OPT}$. For every level $\ell \geq 0$, $G_\ell = (P_\ell, E_\ell)$ is a graph on the representatives P_ℓ . Note that G^* is a Steiner spanner with respect to the point set P_ℓ , as G^* is a spanner on all n points of the input.

We prove Claim 7 using a charging scheme: We charge the weight of every edge in G_ℓ to G^* (more precisely, to line segments along the edges of G^*), and then show that each line segment of weight w in G^* receives $O(\varepsilon^{-3/2} \log \varepsilon^{-1}) \cdot w$ charge. For every point $p \in P_\ell$,

algorithm ALG_1 greedily covers \mathbb{R}^2 by $\Theta(\varepsilon^{-1})$ cones of aperture $\pi/k = \Theta(\varepsilon^{-1})$ and apex p , and adds an edge pq_i in each nonempty cone C_i . For the competitive analysis, we greedily cover \mathbb{R}^2 by $\Theta(\varepsilon^{-1/2})$ cones of aperture $\sqrt{\varepsilon}$ and apex p . We use translates of the same cone cover for all $p \in P_\ell$. Standard volume argument implies that a cone of aperture $\sqrt{\varepsilon}$ intersects $O(\varepsilon^{-1/2})$ cones of aperture $\Theta(\varepsilon^{-1})$. We describe the charging scheme for each such cone \widehat{C} .

Charging Scheme. Consider a cone \widehat{C} with apex p and aperture $\sqrt{\varepsilon}$. Let $E(\widehat{C})$ be the set of edges pq , $q \in \widehat{C}$ that algorithm ALG_1 adds to G_ℓ when p is inserted into P_ℓ . Since \widehat{C} intersects $O(\varepsilon^{-1/2})$ cones of the ordered Yao-graph, then $|E(\widehat{C})| \leq O(\varepsilon^{-1/2})$. By construction, every edge in G_ℓ has weight at most $O(\varepsilon^{-1}2^{-\ell})$. Hence

$$\|E(\widehat{C})\| = \sum_{pq \in E(\widehat{C})} \|pq\| \leq |E(\widehat{C})| \cdot O(\varepsilon^{-1}2^{-\ell}) \leq O(\varepsilon^{-3/2}2^{-\ell}). \quad (3)$$

Let $q_0 = q_0(\widehat{C})$ be a closest point in $P_\ell \cap \widehat{C}$ to p . (Possibly, q_0 arrived after p .) We distinguish between two cases:

Case 1: $\|pq_0\| < 2 \cdot 2^{-\ell}$. Since $q_0 \in P_\ell$, and P_ℓ contains at most one point in each quadtree cell of side length $\Theta(2^{-\ell})$, this case occurs at most $O(1)$ times per apex p . On the one hand, the summation of (3) over all $p \in P_\ell$ and all cones \widehat{C} with $\|pq_0\| < 2 \cdot 2^{-\ell}$ is bounded by $O(|P_\ell| \cdot \varepsilon^{-3/2}2^{-\ell})$. On the other hand, $\text{OPT} \geq \Omega(\|\text{MST}(P_\ell)\|) \geq \Omega(|P_\ell| \cdot 2^{-\ell})$. Consequently, the total weight of all edges handled in Case 1 is $O(\varepsilon^{-3/2}) \text{OPT}$.

Case 2: $\|pq_0\| \geq 2 \cdot 2^{-\ell}$. The optimal spanner G^* contains a pq_0 -path P_0 of weight at most $(1 + \varepsilon)\|pq_0\|$. Recall P_0 lies in the ellipse \mathcal{E}_0 with foci p and q_0 , and $R(p, q_0)$ is the half of \mathcal{E}_0 that contains q_0 (cf. Figure 1). Let $E^*(\widehat{C})$ be the set of maximal line segments e along edges in E^* such that $e \subset P_0 \cap R(p, q_0)$ and $\angle(e, pq_0) \leq 3 \cdot \sqrt{\varepsilon}$. By Lemma 4, we have $\|E^*(\widehat{C})\| \geq \frac{1}{9}\|pq_0\|$. We distribute the weight of all edges in $E(\widehat{C})$ uniformly among the line segments in $E^*(\widehat{C})$. That is, each segment of weight w in $E^*(\widehat{C})$ receives a charge of

$$\frac{\|E(\widehat{C})\|}{\|E^*(\widehat{C})\|} \cdot w \leq \frac{O(\varepsilon^{-3/2}2^{-\ell})}{\Omega(2^{-\ell})} \cdot w \leq O(\varepsilon^{-3/2}) \cdot w. \quad (4)$$

This completes the description of the charging scheme in Case 2.

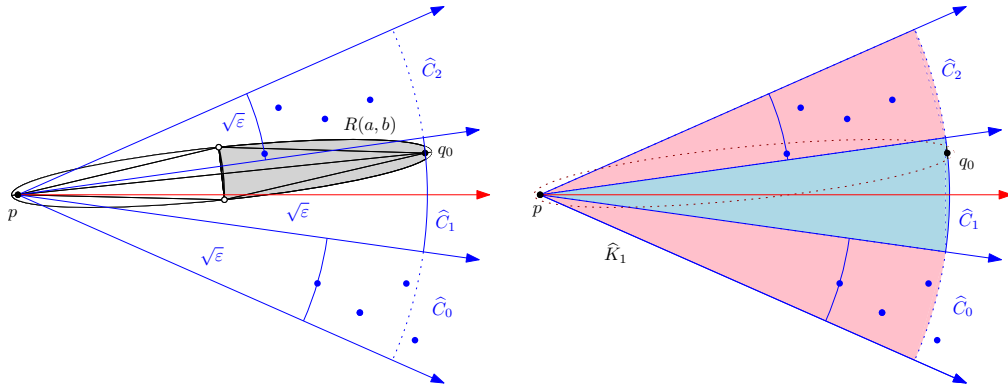
Charges Received. A point along an edge of the optimal spanner G^* may receive charges from several cones \widehat{C} , possibly with different apices $p \in P_\ell$. Let L be a maximal line segment along an edge of G^* such that every point in L receives the same charges.

For a cone \widehat{C} of aperture $\sqrt{\varepsilon}$, let \widehat{K} denote a cone with the same apex and axis as \widehat{C} , but aperture $3\sqrt{\varepsilon}$; refer to Figure 2.

▷ **Claim 8.** If L receives charges from \widehat{C} , then $L \subset \widehat{K}$.

Indeed, if L receive charges from \widehat{C} , then $L \subset R(p, q_0) \subset \mathcal{E}_0$, where \mathcal{E}_0 is the ellipse with foci p and the closest point $q_0 \in \widehat{C} \cap P_\ell$. By Lemma 5, $R(p, q_0)$ lies in a cone with apex p , aperture $2\sqrt{\varepsilon}$, and axis pq_0 . Consequently $L \subset R(p, q_0) \subset \widehat{K}$, which proves Claim 8.

Note that if L receives positive charge from a cone \widehat{C} with apex p and closest point q_0 , then $\angle(L, pq_0) \leq 3 \cdot \sqrt{\varepsilon}$. Since the aperture of the cones \widehat{C} is $\sqrt{\varepsilon}$, then L receives charges from cones \widehat{C} with at most $O(1)$ different orientations. We may restrict ourselves to cones \widehat{C} that are translates of each other (but have different apices in P_ℓ).



■ **Figure 2** Left: Three consecutive cones, \widehat{C}_0 , \widehat{C}_1 , and \widehat{C}_2 , with apex p and aperture $\sqrt{\varepsilon}$. Point q_0 is the closest to p in $P_\ell \cap \widehat{C}_1$; and $R(p, q_0) \subset \widehat{K}_1 = \widehat{C}_0 \cup \widehat{C}_1 \cup \widehat{C}_2$. Right: No point in P_ℓ is in the blue sector \widehat{K} , but there may be points in the pink sectors.

Let \mathcal{A} be the set of all translates of a cone \widehat{C} with aperture $\sqrt{\varepsilon}$ and apices in P_ℓ , and L receives positive charge from \widehat{C} . We partition \mathcal{A} into $O(\log \varepsilon^{-1})$ classes as follows. For $j = 1, \dots, \lceil \log(2\varepsilon^{-1}) \rceil$, let \mathcal{A}_j be the set of cones $\widehat{C} \in \mathcal{A}$ such that $2^{j-\ell} \leq \|pq_0\| < 2^{j+1-\ell}$, where $p \in P_\ell$ is the apex of \widehat{C} and q_0 is the closest point in $P_\ell \cap \widehat{C}$ to p .

▷ **Claim 9.** For each j , segment L receives $O(\varepsilon^{-3/2}) \|L\|$ total charges from all cones in \mathcal{A}_j .

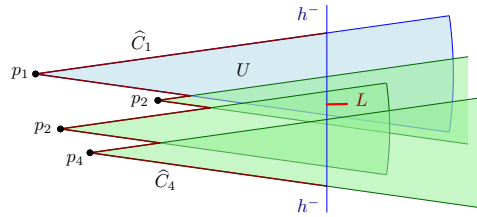
For a cone $\widehat{C} \in \mathcal{A}_j$, the bound (3) is replaced by

$$\|E(\widehat{C})\| = \sum_{pq \in E(\widehat{C})} \|pq\| \leq |E(\widehat{C})| \cdot O(2^{j-\ell}) \leq O(\varepsilon^{-1/2} 2^{j-\ell}), \tag{5}$$

while $\|E^*(\widehat{C})\| \geq \frac{1}{9} \|pq_0\| \geq \Omega(2^{j-\ell})$ by Lemma 4. Refining (4), L receives a charge

$$\frac{\|E(\widehat{C})\|}{\|E^*(\widehat{C})\|} \cdot \|L\| \leq \frac{O(\varepsilon^{-1/2} 2^{j-\ell})}{\Omega(2^{j-\ell})} \cdot \|L\| \leq O(\varepsilon^{-1/2}) \cdot \|L\| \tag{6}$$

from each cone in \mathcal{A}_j . To prove Claim 9, it is enough to show that $|\mathcal{A}_j| \leq O(2^j) \leq O(\varepsilon^{-1})$.



■ **Figure 3** The union U of triangles $\widehat{C} \cap h^-$, where L receives charges from the cones \widehat{C} .

By Claim 8, L received charges from cones of $O(1)$ different orientations. We consider each orientation separately. We may assume w.l.o.g. that the symmetry axis of every cone in \mathcal{A}_j is parallel to the x -axis, and their apex is their leftmost point. Let h be a vertical line that contains the left endpoint of L , and let h^- be the left halfplane bounded by h ; see Figure 3. The intersections $\widehat{C} \cap h$ and $\widehat{K} \cap h$ are vertical line segments of length $O(2^{j-\ell} \tan \sqrt{\varepsilon})$. We have $L \cap h \subset \widehat{K} \cap h$ by Claim 8; and obviously $\widehat{C} \cap h \subset \widehat{K} \cap h$. Consequently, a vertical line segment of length $O(2^{j-\ell} \tan \sqrt{\varepsilon})$ contains $h \cap \widehat{C}$ for all $\widehat{C} \in \mathcal{A}_j$.

18:12 Online Spanners in Metric Spaces

Let U be the union of the triangles $\widehat{C} \cap h^-$ for all $\widehat{C} \in \mathcal{A}_j$. The interior of $\widehat{C} \cap h^-$ does not contain any point in P_ℓ . Consequently, the apices of all cones lie on the boundary ∂U of U . The part of ∂U in h^- is a y -monotone curve with slopes $\pm\sqrt{\varepsilon}$. It follows that the length of ∂U is $O(2^{j-\ell} \tan \sqrt{\varepsilon} / \sin \sqrt{\varepsilon}) = O(2^{j-\ell} \csc \sqrt{\varepsilon}) = O(2^{j-\ell})$. This, in turn, implies that ∂U intersects $O(2^j)$ cubes of side length $a_0 2^{-\ell}$ at level ℓ of the quadtree, and so $|\mathcal{A}_j| \leq O(2^j) \leq O(\varepsilon^{-1})$, as required. This completes the proof Claim 9, and hence the proof of Theorem 6. \blacktriangleleft

3 Lower Bounds in \mathbb{R}^d Under the L_1 Norm

In this section we introduce a strategy based on the points on the integer lattice \mathbb{Z}^d , that achieves a new lower bound for the competitive ratio of an online $(1 + \varepsilon)$ -spanner algorithm in \mathbb{R}^d under the L_1 norm.

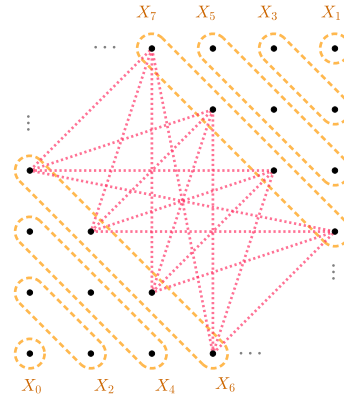


Figure 4 A sketch of the construction for the lower bound in two dimensions. Any online algorithm is required to add the red pairs.

Construction. We describe an adversary strategy with $\Omega_d(\varepsilon^{-d})$ points and show that any online algorithm returns a $(1 + \varepsilon)$ -spanner whose weight is $\Omega_d(\varepsilon^{-d})$ times the optimum weight. One can extend this result for arbitrary number of points, but that does not necessarily improve the lower bound. The final point set X consists of the points of the integer lattice \mathbb{Z}^d in the hypercube $[0, \frac{1}{\varepsilon d}]^d$, where $\varepsilon < \frac{1}{d}$. The points are presented in stages in order to deceive the online algorithm to add more edges than needed. In step $2i$, where $0 \leq i < \frac{1}{2\varepsilon}$, points $x \in X$ such that $\|x\|_1 = i$ will be given to the algorithm. In step $2i + 1$, where $0 \leq i < \frac{1}{2\varepsilon}$, the adversary presents points $x \in X$ such that $\|x\|_1 = \lceil 1/\varepsilon \rceil - i$ (Figure 4). In other words, points are presented in batches according to their L_1 norms.

Competitive Ratio. Denote by X_i the set of points presented in step i . The idea is to show that there has to exist many edges between X_i and X_{i+1} in order to guarantee the $1 + \varepsilon$ stretch-factor. Specifically, we define an *ordered-pair* as follows.

► Definition 10 (ordered-pair). A pair of points (x, y) in \mathbb{R}^d is an ordered-pair if $x \in X_{2i}$ and $y \in X_{2i+1}$ for some i , and $x_k \leq y_k$ for all k , where x_k and y_k are the k -th coordinates of x and y respectively.

Now we show that any ordered-pair $(x, y) \in X_{2i} \times X_{2i+1}$ requires an edge in the spanner immediately after x and y are presented. To prove this, we show (in the full paper) that previously presented points cannot serve as via points in a $(1 + \varepsilon)$ -path between x and y .

► **Lemma 11.** *Let (x, y) be an ordered-pair. Then there is no $(1 + \varepsilon)$ -path between x and y that goes through any other point $z \in X_j$ with $j \leq i + 1$.*

We next show that the total weight of the edges between ordered pairs is $\Omega_d(\varepsilon^{-2d})$.

► **Lemma 12.** *The total weight of the edges between the ordered-pairs is $\Omega_d(\varepsilon^{-2d})$.*

Proof. Let $x = (x_1, \dots, x_d)$ and $y = (y_1, \dots, y_d)$ be two points in X . We show that if $x_k \in [\frac{1}{4\varepsilon(d+0.25)}, \frac{1}{4\varepsilon d}]$ for all $1 \leq k \leq d$, and $y_k \in [\frac{3}{4\varepsilon(d+0.25)}, \frac{3}{4\varepsilon d}]$ for all $1 \leq k \leq d - 1$, then there is a choice of y_d that makes (x, y) an ordered-pair. This would imply that there are $\Omega_d(\varepsilon^{-2d+1})$ ordered-pairs and by Lemma 11, each pair requires an edge of weight $\Omega_d(\varepsilon^{-1})$, thus the total weight of required edges would be $\Omega_d(\varepsilon^{-2d})$.

In order to find such a y_d , recall that $\|x\|_1 + \|y\|_1 = \lceil \varepsilon^{-1} \rceil$ holds because (x, y) is an ordered-pair. This equality uniquely determines the value of y_d ,

$$y_d = \lceil \varepsilon^{-1} \rceil - \sum_{k=1}^d x_k - \sum_{k=1}^{d-1} y_k.$$

We just need to prove the inequalities $y_k \geq x_k$ and $y_k \leq 1/(\varepsilon d)$ for this unique y_k . This can simply be done by plugging the maximum (and minimum) values of x_k s and other y_k s and calculating the result,

$$y_d \geq \frac{1}{\varepsilon} - \frac{d}{4\varepsilon d} - \frac{3(d-1)}{4\varepsilon d} = \frac{3}{4\varepsilon d} > x_d.$$

Also,

$$y_d \leq \frac{1}{\varepsilon} + 1 - \frac{d}{4\varepsilon(d+0.25)} - \frac{3(d-1)}{4\varepsilon(d+0.25)} = 1 + \frac{1}{\varepsilon(d+0.25)} < \frac{1}{\varepsilon d}. \quad \blacktriangleleft$$

Now we can prove the main theorem of this section.

► **Theorem 13.** *The competitive ratio of any online $(1 + \varepsilon)$ -spanner algorithm in \mathbb{R}^d under the L_1 -norm is $\Omega_d(\varepsilon^{-d})$.*

Proof. For the point set $X \subset \mathbb{R}^d$, the unit-distance graph is a Manhattan network: It contains a path of weight $\|xy\|_1$ for all $x, y \in X$. Its weight is $\Theta_d(\varepsilon^{-d})$ which is an upper bound for the weight of a $(1 + \varepsilon)$ -spanner for any $\varepsilon \geq 1$. By Lemma 12, any online algorithm returns a spanner of weight $\Omega_d(\varepsilon^{-2d})$. Thus its competitive ratio is $\Omega_d(\varepsilon^{-d})$. ◀

4 General Metrics: The Ordered Greedy Spanner

In this section we study the online spanners problem on general metric spaces. The points arrive one by one, where for each new point we also receive its distances to all previously introduced points.

In the offline setting, the celebrated greedy spanner algorithm [4] sorts the edges by increasing weight, and then processes them one by one, adding each edge if by the time of examination, the distance between its endpoints is too large. This algorithm achieves

18:14 Online Spanners in Metric Spaces

the existentially optimal² sparsity and lightness as a function of the stretch factor [31]. However, in the online model, we do not receive the edges in a sorted order, and therefore cannot execute the greedy algorithm. As an alternative, we propose here the *ordered greedy* algorithm. This is a deterministic algorithm working against an adaptive adversary. The algorithm receives a stretch factor t , and works naturally as follows: We maintain a spanner H . When a point v_i arrives, we order its edges³ in the original metric by weight. Each edge $\{v_{i'}, v_i\}$ is added to the spanner H if currently $\delta_H(v_{i'}, v_i) > t \cdot \delta_X(v_{i'}, v_i)$. Note that this algorithm can be easily executed in an online fashion.

► **Theorem 14.** *Given an n -point metric space (X, δ_X) in an (adaptive) adversarial order, with stretch factor $t = (2k - 1)(1 + \varepsilon)$ for $k \geq 2$ and $\varepsilon \in (0, 1)$, the ordered greedy algorithm returns a spanner with $O(\varepsilon^{-1} \log \varepsilon^{-1}) \cdot n^{1+\frac{1}{k}}$ edges and weight $O(\varepsilon^{-1} n^{\frac{1}{k}} \log^2 n) \cdot w(\text{MST})$.*

Proof. The bounded stretch of our spanner is straightforward by construction, as every pair was examined at some point, and taken care of. Next we analyze the lightness.

In the online spanning tree problem, points of a finite metric space arrive one-by-one, and we need to connect each new point to a previous point to maintain a spanning tree. The ordered greedy algorithm connects each vertex v_i , to the closest vertex in $\{v_1, \dots, v_{i-1}\}$. As was shown by Imase and Waxman [44], the tree created by the ordered greedy algorithm has lightness $O(\log n)$, which is the best possible [44]. Denote the online spanning tree by T_G . Note that the ordered greedy spanner H will contain T_G , as a shortest edge between a new vertex to a previously introduced vertex is always added to the spanner H . The following clustering lemma is frequently used for spanner constructions (see e.g. [3, 22, 27]). We provide a proof for the sake of completeness.

▷ **Claim 15.** For every $i \in \mathbb{N}$, the point set X can be partitioned into clusters \mathcal{C}_i of diameter at most $D_i = \varepsilon \cdot (1 + \varepsilon)^i$ w.r.t. the metric δ_{T_G} such that $|\mathcal{C}_i| = O\left(\frac{w(T_G)}{\varepsilon \cdot (1 + \varepsilon)^i}\right)$.

Proof. Let N_i be a maximal set of vertices such that for every $x, y \in N_i$, $\delta_{T_G}(x, y) > \frac{1}{2} \cdot D_i$. For every vertex $x \in N_i$ let $C_x = \{z : x = \operatorname{argmin}_{y \in N_i} \delta_X(z, y)\}$ be the Voronoi cell of x . Clearly, $\operatorname{diam}(C_x) \leq D_i$ for all x . Further, consider a continuous version of T_G (where each edge is an interval). Then as the graph T_G is connected, each cluster C_x contains at least $\frac{1}{4} D_i$ length of edges (as the balls $\{B_{T_G}(x, \frac{1}{4} D_i)\}_{x \in N_i}$ are pairwise disjoint). It follows that

$$|\mathcal{C}_i| = |N_i| \leq \frac{w(T_G)}{\frac{1}{4} D_i} = O\left(\frac{w(T_G)}{\varepsilon \cdot (1 + \varepsilon)^i}\right). \quad \triangleleft$$

For every i , consider the *scale* $E_i = \{e = \{u, v\} \in H : (1 + \varepsilon)^{i-1} \leq \delta_X(u, v) < (1 + \varepsilon)^i\}$. We are now ready to bound the lightness and the sparsity of the ordered greedy spanner. This is accomplished in the next two claims, with proofs in the full paper.

▷ **Claim 16.** The weight of the ordered greedy spanner is $O(n^{\frac{1}{k}} \cdot \varepsilon^{-2} \log^2 n) \cdot w(\text{MST})$.

▷ **Claim 17.** The ordered greedy spanner has $O(\varepsilon^{-1} \log \frac{1}{\varepsilon}) \cdot n^{1+\frac{1}{k}}$ edges.

This completes the proof of Theorem 14. ◀

² Specifically, if a t -spanner construction achieves an upper bound $m(n, t)$ and $l(n, t)$, resp., on the size and lightness of an n -vertex graph then this bound also holds for the greedy t -spanner [31].

³ By edges we mean point pairs in the metric space, we will often use notation from graph theory.

5 Lower Bound for General Metrics

In this section we prove an $\Omega(\frac{1}{k} \cdot n^{\frac{1}{k}})$ lower bound on the competitive ratio of an online $(2k - 1)$ -spanner of n -vertex graphs. Our lower bound holds in both cases where the quality is measured by number of edges or the weight. It follows that our upper bound in Theorem 14 cannot be substantially improved, even if we consider competitive ratio instead of lightness/sparsity.

Recall that the Erdős Girth Conjecture [28] states that for every $n, k \geq 1$, there exists an n -vertex graph with $\Omega(n^{1+\frac{1}{k}})$ edges and girth $2k + 2$. The proof of the following lemma is based on a counting argument from the recent lower bound proof for (static) vertex fault tolerant emulators by Bodwin, Dinitz, and Nazari [15].

► **Lemma 18.** *Assuming the Erdős girth conjecture, for every $n, k \geq 1$, there exists an n -point metric space (X, δ_X) with diameter $2k - 1$, such that every $(2k - 1)$ -spanner has $\Omega(\frac{1}{k} \cdot n^{1+\frac{1}{k}})$ edges and weight $\Omega(n^{1+\frac{1}{k}})$.*

Proof. Let $G = (V, E_G)$ be the graph fulfilling the Erdős girth conjecture. That is, G is an unweighted n -vertex graph with girth $2k + 2$ and $|E_G| = \Omega(n^{1+\frac{1}{k}})$ edges. Set a metric δ_X over V as follows,⁴

$$\forall u, v \in V \quad \delta_X(u, v) = \min \{ \delta_G(u, v), 2k - 1 \}.$$

Suppose that $H = (V, E_H)$ is a $(2k - 1)$ -spanner for (V, δ_X) with weight function w_H , where the weight of an edge $e' \in \{u, v\} \in E_H$ is $w_H(e') = \delta_X(u, v)$. Let $E' = E_H \setminus E_G$ be the edges of H which are not in G . We say that an edge $e' \in E'$ covers an edge $e \in E_G$, if there is a shortest path in G between the endpoints of e' going through e of weight at most k . Note that as e' has weight at most k , there is a unique shortest path in G between its endpoints. In particular, each edge $e \in E'$ can cover at most k edges in E_G .

Consider an edge $e = \{v_0, v_s\} \in E_G \setminus E_H$. We argue that some edge $e' \in E'$ must cover e . Suppose for contradiction otherwise, and let $P = (v_0, v_1, \dots, v_s)$ be the shortest path in H between the endpoints v_0, v_s of e . Suppose first that P contains an edge v_i, v_{i+1} of weight at least $w_H(\{v_i, v_{i+1}\}) \geq k + 1$. In particular, $\delta_G(\{v_i, v_{i+1}\}) \geq k + 1$. Then by the triangle inequality, $\delta_G(v_0, v_i) + \delta_G(v_{i+1}, v_s) \geq \delta_G(v_i, v_{i+1}) - \delta_G(v_0, v_s) \geq k$. It follows that P has weight at least $2k + 1$, a contradiction to the fact that H is a $2k - 1$ spanner. We conclude that for every $i \in \{0, \dots, s - 1\}$, $\delta_X(v_i, v_{i+1}) = \delta_G(v_i, v_{i+1}) \leq k$. In particular, in G there is a unique path $P_i = (u_0^i, \dots, u_{s_i}^i)$ between v_i to v_{i+1} of weight $\delta_G(v_i, v_{i+1}) \leq k$. As no edge covers e , e does not belong to any of these paths. The concatenation of these paths $P_0 \circ P_1 \circ \dots \circ P_{s-1}$ is a path in G of at most $2k - 1$ edges between the endpoints of e . It follows that G contains a $2k$ -cycle, a contradiction.

For conclusion, as every edge in $E_G \setminus E_H$ is covered, and every edge in $E' = E_H \setminus E_G$ can cover at most k edges, it follows that $|E_H \setminus E_G| \geq \frac{1}{k} \cdot |E_G \setminus E_H|$. In particular,

$$|E_H| = |E_H \cap E_G| + |E_H \setminus E_G| \geq |E_H \cap E_G| + \frac{1}{k} \cdot |E_G \setminus E_H| \geq \frac{1}{k} \cdot |E_G|.$$

To bound the weight, for each edge $e' = \{s, t\} \in E'$, let $A_{e'}$ be the set of edges in E_G covered by e' . Note that $w_H(e') = \delta_G(s, t) = |A_{e'}|$. As all the edges in $E_G \setminus E_H$ are covered, we conclude

⁴ Note that $\forall x, y, z \in V$, $\delta_X(x, z) = \min \{ \delta_G(x, z), 2k - 1 \} \leq \min \{ \delta_G(x, y) + \delta_G(y, z), 2k - 1 \} \leq \min \{ \delta_G(x, y), 2k - 1 \} + \min \{ \delta_G(y, z), 2k - 1 \} = \delta_X(x, y) + \delta_X(y, z)$. Thus δ_X is a metric space.

$$\begin{aligned}
w_H(E_H) &= w_H(E_H \cap E_G) + w_H(E_H \setminus E_G) \\
&= |E_H \cap E_G| + \sum_{e' \in E'} |A_{e'}| \\
&\geq |E_H \cap E_G| + |E_G \setminus E_H| = |E_G| = \Omega(n^{1+\frac{1}{k}}). \quad \blacktriangleleft
\end{aligned}$$

► **Theorem 19.** *Assuming Erdős girth conjecture, the competitive ratio of any online $(2k-1)$ -spanner algorithm for n -point metrics is $\Omega(\frac{1}{k} \cdot n^{\frac{1}{k}})$, for both weight and number of edges. In more details, there is an n -point metric space (X, δ_X) with a $(2k-1)$ -spanner $H_{\text{OPT}} = (X, E_{\text{OPT}})$, and order over X for which every $(2k-1)$ -spanner produced by an online algorithm will have $\Omega(\frac{1}{k} \cdot n^{\frac{1}{k}}) \cdot |E_{\text{OPT}}|$ edges, and $\Omega(\frac{1}{k} \cdot n^{\frac{1}{k}}) \cdot w(H_{\text{OPT}})$ weight.*

Proof. Consider the metric space (X, δ_X) from Lemma 18 with parameters $n-1$ and k . Let X' be the metric space X with an additional point r at distance $\frac{2k-1}{2}$ from all the points in X . Note that no pairwise distance is changed due to the introduction of r . The adversary provides the online algorithm the points in X first (in some arbitrary order), and the point r last. After the algorithm received all the points in X' , it has a $2k-1$ -spanner H_{n-1} . According to Lemma 18, H_{n-1} has $\Omega(\frac{1}{k} \cdot (n-1)^{1+\frac{1}{k}}) = \Omega(\frac{1}{k} \cdot n^{1+\frac{1}{k}})$ edges, and $\Omega(n^{1+\frac{1}{k}})$ weight.

Next the algorithm introduces r . Consider the spanner $S = (X', E_S)$ consisting of $n-1$ edges with r as a center. Note that the maximum distance in S is $2k-1$, and hence S is a $2k-1$ spanner as required. Note that S contains $n-1$ edges of weight $\frac{2k-1}{2}$ each, and thus have total weight of $O(nk)$. We conclude

$$\begin{aligned}
|E_{H_n}| &\geq |E_{H_{n-1}}| = \Omega(\frac{1}{k} \cdot n^{1+\frac{1}{k}}) = \Omega(\frac{1}{k} \cdot n^{\frac{1}{k}}) \cdot |E_S|. \\
w(E_{H_n}) &\geq w(E_{H_{n-1}}) = \Omega(n^{1+\frac{1}{k}}) = \Omega(\frac{1}{k} \cdot n^{\frac{1}{k}}) \cdot w(S). \quad \blacktriangleleft
\end{aligned}$$

6 Conclusion

We studied online spanners for points in metric spaces. In the Euclidean d -space, we presented an online $(1+\varepsilon)$ -spanner algorithm with competitive ratio $O(\varepsilon^{1-d} \log n)$, improving the previous bound of $O_d(\varepsilon^{-(d+1)} \log n)$ from [14]. In fact, the spanner maintained by the algorithm has $O_d(\varepsilon^{1-d} \log \varepsilon^{-1}) \cdot n$ edges, almost matching the (offline) optimal bound of $O_d(\varepsilon^{1-d}) \cdot n$. Moreover, in the plane, a tighter analysis of the same algorithm provides an almost quadratic improvement of the competitive ratio to $O(\varepsilon^{-3/2} \log \varepsilon^{-1} \log n)$, by comparing the online spanner with an instance-optimal spanner directly, circumventing the comparison to an MST (i.e., lightness). Note that, the logarithmic dependence on n is unavoidable due to a $\Omega((\varepsilon^{-1}/\log \varepsilon^{-1}) \log n)$ lower bound in the real line [14]. However, our lower bound $\Omega(\varepsilon^{-d})$ under the L_1 -norm in \mathbb{R}^d shows a dependence on the dimension. This leads to the following question.

► **Question.** *Does the competitive ratio of an online $(1+\varepsilon)$ -spanning algorithm for n points in \mathbb{R}^d necessarily grow proportionally with $\varepsilon^{-f(d)} \cdot \log n$, where $\lim_{d \rightarrow \infty} f(d) = \infty$?*

Interestingly, for $t \in [(1+\varepsilon)\sqrt{2}, (1-\varepsilon)2]$, we can show that every online t -spanner algorithm in \mathbb{R}^d must have competitive ratio $2^{\Omega(\varepsilon^{2d})}$ (see the full paper for further details).

Next, we studied online spanners in general metrics. We showed that the *ordered greedy* algorithm maintains a spanner with $O(\varepsilon^{-1} \log \varepsilon^{-1}) \cdot n^{1+\frac{1}{k}}$ edges and $O(\varepsilon^{-1} n^{\frac{1}{k}} \log^2 n)$ lightness, with stretch factor $t = (2k-1)(1+\varepsilon)$ for $k \geq 2$ and $\varepsilon \in (0, 1)$, for a sequence of n points in

a metric space. Moreover, we show that these bounds cannot be significantly improved, by introducing an instance that achieves an $\Omega(\frac{1}{k} \cdot n^{1/k})$ competitive ratio on both sparsity and lightness. Finally, we established the trade-off among stretch, number of edges and lightness for points in ultrametrics, showing that one can maintain a $(2 + \varepsilon)$ -spanner for ultrametrics with $O(\varepsilon^{-1} \log \varepsilon^{-1}) \cdot n$ edges and $O(\varepsilon^{-2})$ lightness.

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. doi:10.1145/1198513.1198522.
- 2 Noga Alon and Yossi Azar. On-line Steiner trees in the Euclidean plane. *Discrete & Computational Geometry*, 10:113–121, 1993. doi:10.1007/BF02573969.
- 3 Stephen Alstrup, Søren Dahlgaard, Arnold Filtser, Morten Stöckel, and Christian Wulff-Nilsen. Constructing light spanners deterministically in near-linear time. *Theoretical Computer Science*, 2022. doi:10.1016/j.tcs.2022.01.021.
- 4 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.
- 5 Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized Steiner problem. *Theoretical Computer Science*, 324(2-3):313–324, 2004. doi:10.1016/j.tcs.2004.05.021.
- 6 Surender Baswana. Streaming algorithm for graph spanners—single pass and constant processing time per edge. *Inf. Process. Lett.*, 106(3):110–114, 2008. doi:10.1016/j.ipl.2007.11.001.
- 7 Surender Baswana, Sumeet Khurana, and Soumojit Sankar. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms*, 8(4):35:1–35:51, 2012. doi:10.1145/2344422.2344425.
- 8 Ruben Becker, Sebastian Forster, Andreas Karrenbauer, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *SIAM J. Comput.*, 50(3):815–856, 2021. doi:10.1137/19M1286955.
- 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. 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1836–1855, 2021. doi:10.1137/1.9781611976465.110.
- 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. doi:10.1145/258533.258618.
- 11 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In *Proc. 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1899–1918, 2019. doi:10.1137/1.9781611975482.115.
- 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. doi:10.4230/LIPICs.SoCG.2021.15.
- 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. doi:10.4230/LIPICs.STACS.2021.13.
- 14 Sujoy Bhore and Csaba D. Tóth. Online Euclidean spanners. In *Proc. 29th European Symposium on Algorithms (ESA)*, volume 204 of *LIPICs*, pages 116:1–16:19. Schloss Dagstuhl, 2021. doi:10.4230/LIPICs.ESA.2021.16.
- 15 Greg Bodwin, Michael Dinitz, and Yasamin Nazari. Vertex fault-tolerant emulators. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *LIPICs*, pages 25:1–25:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl. doi:10.4230/LIPICs.ITCS.2022.25.

- 16 Greg Bodwin and Sebastian Krinninger. Fully dynamic spanners with worst-case update time. In *Proc. 24th Annual European Symposium on Algorithms (ESA)*, volume 57 of *LIPICs*, pages 17:1–17:18. Schloss Dagstuhl, 2016. doi:10.4230/LIPICs.ESA.2016.17.
- 17 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998. See this url.
- 18 Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Greedy spanners are optimal in doubling metrics. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms SODA*, pages 2371–2379, 2019. doi:10.1137/1.9781611975482.145.
- 19 Prosenjit Bose, Jean-Lou De Carufel, Pat Morin, André van Renssen, and Sander Verdonschot. Towards tight bounds on theta-graphs: More is not always better. *Theor. Comput. Sci.*, 616:70–93, 2016. doi:10.1016/j.tcs.2015.12.017.
- 20 Prosenjit Bose, Joachim Gudmundsson, and Pat Morin. Ordered theta graphs. *Computational Geometry*, 28(1):11–18, 2004. doi:10.1016/j.comgeo.2004.01.003.
- 21 Paul B. Callahan and S. Rao Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 291–300, 1993. URL: <http://dl.acm.org/citation.cfm?id=313559.313777>.
- 22 Shiri Chechik and Christian Wulff-Nilsen. Near-optimal light spanners. *ACM Trans. Algorithms*, 14(3):33:1–33:15, 2018. doi:10.1145/3199607.
- 23 L. Paul Chew. There are planar graphs almost as good as the complete graph. *J. Comput. Syst. Sci.*, 39(2):205–219, 1989. doi:10.1007/BF01758846.
- 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. doi:10.1145/28395.28402.
- 25 Michael J. Demmer and Maurice P. Herlihy. The arrow distributed directory protocol. In *Proc. 12th Symposium on Distributed Computing (DISC)*, volume 1499 of *LNCS*, pages 119–133. Springer, 1998. doi:10.1007/BFb0056478.
- 26 Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Trans. Algorithms*, 7(2):20:1–20:17, 2011. doi:10.1145/1921659.1921666.
- 27 Michael Elkin and Shay Solomon. Fast constructions of lightweight spanners for general graphs. *ACM Trans. Algorithms*, 12(3):29:1–29:21, 2016. doi:10.1145/2836167.
- 28 P. Erdős. Extremal problems in graph theory. *Theory of Graphs and Its Applications (Proc. Sympos. Smolenice)*, pages 29–36, 1964. see here.
- 29 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, 2008. doi:10.1137/070683155.
- 30 Arnold Filtser, Michael Kapralov, and Navid Nouri. Graph spanners by sketching in dynamic streams and the simultaneous communication model. In *Proc. 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1894–1913, 2021. doi:10.1137/1.9781611976465.113.
- 31 Arnold Filtser and Shay Solomon. The greedy spanner is existentially optimal. *SIAM J. Comput.*, 49(2):429–447, 2020. doi:10.1137/18M1210678.
- 32 John Fischer and Sarel Har-Peled. Dynamic well-separated pair decomposition made easy. In *Proc. 17th Canadian Conference on Computational Geometry (CCCG)*, pages 235–238, 2005. see here. URL: <http://www.cccg.ca/proceedings/2005/32.pdf>.
- 33 Jie Gao, Leonidas J. Guibas, and An Nguyen. Deformable spanners and applications. *Comput. Geom.*, 35(1-2):2–19, 2006. doi:10.1016/j.comgeo.2005.10.001.
- 34 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. doi:10.1109/TIT.2017.2713820.
- 35 Lee-Ad Gottlieb and Liam Roditty. An optimal dynamic spanner for doubling metric spaces. In *Proc. 16th Annual European Symposium on Algorithms (ESA)*, volume 5193 of *LNCS*, pages 478–489. Springer, 2008. doi:10.1007/978-3-540-87744-8_40.

- 36 Elena Grigorescu, Young-San Lin, and Kent Quanrud. Online directed spanners and Steiner forests. In *Proc. Approximation, Randomization, and Combinatorial Optimization Algorithms and Techniques (APPROX/RANDOM)*, volume 207 of *LIPIcs*, pages 5:1–5:25. Schloss Dagstuhl, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.5.
- 37 Joachim Gudmundsson and Christian Knauer. Dilation and detours in geometric networks. In *Handbook of Approximation Algorithms and Metaheuristics*, volume 2. Chapman and Hall/CRC, 2nd edition, 2018. see here.
- 38 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. doi:10.1145/1328911.1328921.
- 39 Anupam Gupta, R. Ravi, Kunal Talwar, and Seeun William Umboh. LAST but not least: Online spanners for buy-at-bulk. In Philip N. Klein, editor, *Proc. 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 589–599, 2017. doi:10.1137/1.9781611974782.38.
- 40 MohammadTaghi Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Online node-weighted steiner forest and extensions via disk paintings. *SIAM J. Comput.*, 46(3):911–935, 2017. doi:10.1137/14098692X.
- 41 Sarel Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Mathematical Surveys and Monographs*. AMS, Providence, RI, 2011. see here.
- 42 Sarel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006. doi:10.1137/S0097539704446281.
- 43 Maurice Herlihy, Srikanta Tirthapura, and Rogert Wattenhofer. Competitive concurrent distributed queuing. In *Proc. 20th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 127–133, 2001. doi:10.1145/383962.384001.
- 44 Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991. doi:10.1137/0404033.
- 45 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. doi:10.1007/3-540-19487-8_23.
- 46 Robert Krauthgamer and James R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proc 15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 798–807, 2004. see here. URL: <http://dl.acm.org/citation.cfm?id=982792.982913>.
- 47 Hung Le and Shay Solomon. Truly optimal Euclidean spanners. In *Proc. 60th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1078–1100, 2019. doi:10.1109/FOCS.2019.00069.
- 48 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. doi:10.4230/LIPIcs.ESA.2020.67.
- 49 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.
- 50 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. doi:10.1109/FOCS.2011.65.
- 51 Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007. doi:10.1017/CB09780511546884.
- 52 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.
- 53 David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989. doi:10.1137/0218050.
- 54 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM (JACM)*, 36(3):510–530, 1989. doi:10.1145/65950.65953.

- 55 Liam Roditty. Fully dynamic geometric spanners. *Algorithmica*, 62(3-4):1073–1087, 2012. doi:10.1007/s00453-011-9504-7.
- 56 Jim Ruppert and Raimund Seidel. Approximating the d -dimensional complete Euclidean graph. In *Proc. 3rd Canadian Conference on Computational Geometry (CCCG)*, pages 207–210, 1991.
- 57 Christian Schindelhauer, Klaus Volbert, and Martin Ziegler. Geometric spanners with applications in wireless networks. *Computational Geometry*, 36(3):197–214, 2007. doi:10.1016/j.comgeo.2006.02.001.
- 58 Michiel H. M. Smid. The well-separated pair decomposition and its applications. In *Handbook of Approximation Algorithms and Metaheuristics*, volume 2. CRC Press, 2nd edition, 2018. URL: <https://www.taylorfrancis.com/chapters/edit/10.1201/9781351235426-4/well-separated-pair-decomposition-applications-michiel-smid>.
- 59 Seeun William Umboh. Personal communication, October 2021.
- 60 Andrew Chi-Chih Yao. Space-time tradeoff for answering range queries (extended abstract). In *Proc. 14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 128–136, 1982. doi:10.1145/800070.802185.

Sparse Temporal Spanners with Low Stretch

Davide Bilò  

Department of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, Italy

Gianlorenzo D'Angelo  

Gran Sasso Science Institute, L'Aquila, Italy

Luciano Gualà  

Department of Enterprise Engineering, University of Rome "Tor Vergata", Italy

Stefano Leucci  

Department of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, Italy

Mirko Rossi  

Gran Sasso Science Institute, L'Aquila, Italy

Abstract

A *temporal graph* is an undirected graph $G = (V, E)$ along with a function $\lambda : E \rightarrow \mathbb{N}^+$ that assigns a time-label to each edge in E . A path in G such that the traversed time-labels are non-decreasing is called a *temporal path*. Accordingly, the distance from u to v is the minimum length (i.e., the number of edges) of a temporal path from u to v . A temporal α -spanner of G is a (temporal) subgraph H that preserves the distances between any pair of vertices in V , up to a *multiplicative* stretch factor of α . The *size* of H is measured as the number of its edges.

In this work, we study the size-stretch trade-offs of temporal spanners. In particular we show that temporal cliques always admit a temporal $(2k - 1)$ -spanner with $\tilde{O}(kn^{1+\frac{1}{k}})$ edges, where $k > 1$ is an integer parameter of choice. Choosing $k = \lfloor \log n \rfloor$, we obtain a temporal $O(\log n)$ -spanner with $\tilde{O}(n)$ edges that has almost the same size (up to logarithmic factors) as the temporal spanner given in [Casteigts et al., JCSS 2021] which only preserves temporal connectivity.

We then turn our attention to *general* temporal graphs. Since $\Omega(n^2)$ edges might be needed by any connectivity-preserving temporal subgraph [Axiotis et al., ICALP'16], we focus on approximating distances from a *single source*. We show that $\tilde{O}(n/\log(1 + \varepsilon))$ edges suffice to obtain a stretch of $(1 + \varepsilon)$, for any small $\varepsilon > 0$. This result is essentially tight in the following sense: there are temporal graphs G for which any temporal subgraph preserving exact distances from a single-source must use $\Omega(n^2)$ edges. Interestingly enough, our analysis can be extended to the case of *additive* stretch for which we prove an upper bound of $\tilde{O}(n^2/\beta)$ on the size of any temporal β -additive spanner, which we show to be tight up to polylogarithmic factors.

Finally, we investigate how the *lifetime* of G , i.e., the number of its distinct time-labels, affects the trade-off between the size and the stretch of a temporal spanner.

2012 ACM Subject Classification Theory of computation \rightarrow Sparsification and spanners

Keywords and phrases temporal spanners, temporal graphs, graph sparsification, approximate distances

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.19

Related Version *Full Version:* <https://arxiv.org/abs/2206.11113>

Funding *Davide Bilò:* Partially supported by the research project "Spanners and Oracles in Static and Temporal Networks" (04ATE2022.AVVIO.Bilò) funded by the University of L'Aquila.

Gianlorenzo D'Angelo: Partially supported by the Italian MIUR PRIN 2017 Project ALGADIMAR "Algorithms, Games, and Digital Markets".

Stefano Leucci: Partially supported by the research project "Spanners and Oracles in Static and Temporal Networks" (04ATE2022.AVVIO.Bilò) funded by the University of L'Aquila.



© Davide Bilò, Gianlorenzo D'Angelo, Luciano Gualà, Stefano Leucci, and Mirko Rossi;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 19; pp. 19:1–19:16

Leibniz International Proceedings in Informatics



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

1 Introduction

A *temporal graph* is a graph $G = (V, E)$ in which each edge can be used only in certain time instants. This recurrent idea of time-evolving graphs has been formalized in multiple ways, and a simple widely-adopted model is the one of Kempe, Kleinberg, and Kumar [11], in which each edge $e \in E$ has an assigned time-label $\lambda(e)$ representing the instant in which e can be used. A path from a vertex to another in G is said to be a *temporal path* if the time-labels of the traversed edges are non-decreasing. Accordingly, a graph is *temporally connected* if there exists a temporal path from u to v , for every two vertices $u, v \in V$.

Notice that, unlike paths in *static* graphs, the existence of temporal paths is neither symmetric nor transitive.¹ For this reason, temporal graphs exhibit a different combinatorial structure compared to static graphs, and even problems that admit easy solutions on static graphs become more challenging in their temporal counterpart. Indeed, one of the main problems introduced in the seminal paper of Kempe, Kleinberg, and Kumar [11] is that of finding a sparse temporally connected subgraph H of an input temporal graph G . Such a subgraph H is sometimes referred to as a *temporal spanner* of G . While any spanning-tree is trivially a connectivity-preserving subgraph of a *static* graph, not all temporal graphs G admit a temporal spanner having $O(n)$ edges [11]. In particular, [11] exhibits a class of temporal graphs that contain $\Theta(n \log n)$ edges and cannot be further sparsified. Later, [4] provided a stronger negative result showing that there are temporal graphs G such that any temporal spanner of G must use $\Theta(n^2)$ edges. These strong lower bounds on general graphs motivated [7] to focus on *temporal cliques* instead. Here the situation improves significantly, as only $O(n \log n)$ edges are sufficient to guarantee temporal connectivity. This gives rise to the following natural question, which is exactly the focus of our paper: *can one design a temporal spanner that also guarantees short temporal paths between any pair of vertices?*

To address this question, we measure the *length* of a temporal path as the number of its edges,² and we introduce the notion of temporal α -spanner of a temporal graph G , i.e., a subgraph H of G such that $d_H(u, v) \leq \alpha \cdot d_G(u, v)$ for every pair of vertices $u, v \in V$, where $d_H(u, v)$ (resp. $d_G(u, v)$) denotes the length of a shortest temporal path from u to v in H (resp. G). Our main question then becomes that of understanding which trade-offs can be achieved between the *size*, i.e., the number of edges, of H and the value of its *stretch-factor* α . This same question received considerable attention on static graphs and gave rise to a significant amount of work (see, e.g., [1]), hence we deem investigating its temporal counterpart as a very interesting research direction.

To the best of our knowledge, the only temporal α -spanner currently known is actually the connectivity-preserving subgraph of [7] having size $O(n \log n)$. However, a closer inspection of its construction shows that the resulting α -spanner can have stretch $\alpha = \Theta(n)$. In particular, even the problem of achieving stretch $o(n)$ using $o(n^2)$ edges remains open.

In this paper we investigate which size-stretch trade-offs can be attained by selecting subgraphs of temporal graphs, as detailed in the following.

¹ Indeed, a temporal path from u to v is not necessarily a temporal path from v to u , even when G is undirected. Moreover, the existence of a temporal path from u to v , and of a temporal path from v to w , does not imply the existence of a temporal path from u to w .

² Alternative definitions for the length of a temporal path are also natural, e.g., the *arrival time*, *departure time*, *duration*, or *travel time* as we briefly discuss in the conclusions.

1.1 Our results

Temporal cliques. Following [7], we start by considering temporal cliques (see Section 3). Our main result is the following: given a temporal clique G and an integer $k \geq 2$, we can construct, in polynomial time, a temporal $(2k-1)$ -spanner of G having size $O(kn^{1+1/k} \log^{1-1/k} n)$. Interestingly, the special case $k = \lfloor \log n \rfloor$ shows that $O(n \log^2 n)$ edges suffice to ensure that a temporal path of length $O(\log n)$ exists between any pair of vertices. For this choice of k , the size of our spanner is only a logarithmic factor away from the size the temporal spanner of [7] that uses $O(n \log n)$ edges and only preserves connectivity. We obtain our results by constructing hierarchical clustering of the vertices that guides the constructions of temporal paths.

We also show that there are temporal cliques for which any temporal spanner with stretch smaller than 3 must have $\Omega(n^2)$ edges.

Single-source temporal spanners on general graphs. Next, in Section 4, we move our attention from temporal cliques to general temporal graphs. As already pointed out, there are temporal graphs that do not admit any connectivity-preserving subgraph with $o(n^2)$ edges [4]. Hence, we consider the special case in which we have a single source s . One can observe that any temporal graph G admits a temporal subgraph containing $O(n)$ edges and preserving the connectivity from s (see also [11]). However, to the best of our knowledge, no non-trivial result is known on the size of subgraphs preserving approximate distances from s .

We formalize this problem by introducing the notion of *single-source temporal α -spanner* of $G = (V, E)$ w.r.t. a source $s \in V$, which we define as a subgraph H of G such that $d_H(s, v) \leq \alpha \cdot d_G(s, v)$ for every $v \in V$. Our main contribution for the single-source case is the following: given any temporal graph G , we can compute in polynomial time a single-source temporal $(1 + \varepsilon)$ -spanner having size $O(\frac{n \log^4 n}{\log(1+\varepsilon)})$, where $\varepsilon > 0$ is a parameter of choice.

Furthermore, we show that any single-source temporal 1-spanner (i.e., a subgraph preserving *exact* distances from s) must have $\Omega(n^2)$ edges in general. Our construction can be generalized to provide a lower bound of $\Omega(\frac{n^2}{\beta})$ on the size of any single-source temporal β -additive spanner, namely a subgraph H that preserves single-source distances up to an *additive* term of at most $\beta \geq 1$ (i.e., we require $d_H(s, v) \leq d_G(s, v) + \beta$ for all $v \in V$).

Interestingly, the same techniques used to obtain our single-source temporal $(1 + \varepsilon)$ -spanner can be also applied to build a single-source temporal β -additive spanner of size $O(\frac{n^2 \log^4 n}{\beta})$, which essentially matches our aforementioned lower bound.

The role of lifetime. An important parameter that measures how time-dependent is a temporal graph $G = (V, E)$ is its *lifetime*, i.e., the number L of distinct time-labels associated with the edges of G . Indeed, a temporal graph with lifetime $L = 1$ is just a *static* graph, while any temporal graph trivially satisfies $L = O(n^2)$. It is not surprising that the lifetime plays a crucial role in determining the number of edges required by temporal spanners. For example, the lower bound of $\Omega(n^2)$ on the size of any connectivity-preserving temporal subgraph requires $L = \Omega(n)$ [4]. In this paper, we also present a collection of results with the goal of shedding some light on the lifetime-size trade-off of temporal spanners. In particular, our results provide the following lifetime-dependant upper bounds on the size of temporal α -spanners

- As far as temporal cliques are concerned, we show how to build, in polynomial time, a temporal 3-spanner with $O(2^L n \log n)$ edges. This implies that, when $L = O(1)$, we can achieve stretch 3 with $\tilde{O}(n)$ edges.³

³ The notation $\tilde{O}(f(n))$ is a synonym for $O(f(n) \cdot \text{polylog } f(n))$.

- If $L = 2$, we can find (in polynomial time) a temporal 2-spanner of a temporal clique having size $O(n \log n)$. We deem this result interesting since, as soon as $L > 2$, our lower bound of $\Omega(n^2)$ on the size of any temporal 2-spanner still applies.
- We show that, when L is small, *general* temporal graphs can be sparsified by exploiting known size-stretch trade-offs for spanners of static graphs. In particular, we show that if it is possible to compute, in polynomial time, an α -spanner of a static graph having size $f(n)$, then one can also build a temporal α -spanner of size $O(Lf(n))$. This yields, e.g., a temporal $\lfloor \log n \rfloor$ -spanner of size $o(n^2)$ on general temporal graphs with $L = o(n)$.

Due to space limitations, these results and some of the proofs are omitted and can be found in the full version of the paper.

1.2 Related work

The definitions of temporal graphs and temporal paths given in the literature sometimes differ from the ones we adopt here. We now discuss how our results relate to some of the most common variants. A first difference concerns the notion of temporal paths: some authors consider *strict temporal paths* [2, 7, 11], i.e., temporal paths in which edge labels must be strictly increasing (rather than non-decreasing). As observed by [11], if we adopt strict temporal paths then there are dense graphs that cannot be sparsified, indeed no edge can be removed from a temporal clique in which all edges have the same time-label. As observed in [7], one can get rid of these problematic instances by assuming that time-labels are *locally distinct*, namely that all the time-labels of the edges incident to any single vertex are distinct. In this case all temporal paths are also strict temporal paths and hence they focus on temporal paths as defined in our paper. A second difference concerns whether edges are allowed to have multiple time-labels, as in [2, 12]. In this case, each edge e is associated to a non-empty set of time instants $\lambda(e) \subseteq \mathbb{N}^+$ in which e is available. We observe that any algorithm that sparsifies a temporal clique with single time-labels can be directly used on the case of multiple time-labels by selecting an arbitrary time-label for each edge (see also the discussion in [7]). This is no longer true when we consider general temporal graphs, since removing edge labels might affect distances. However, all our algorithms work also in the case of multiple labels and, since our lower bounds are given for single labels, they also apply to the case of multiple labels.

Another research line concerns random temporal graphs. In particular, temporal cliques in which each edge has a single time-label chosen u.a.r. from the set $\{1, \dots, \alpha\}$, where $\alpha \geq 4$, admit temporal spanners with $O(n \log n)$ edges w.h.p. [2]. In [8], the authors study connectivity properties of random temporal graphs defined as an Erdős-Rényi graph $G_{n,p}$ in which each edge e has time-label chosen as the rank of e in a random permutation of the graph's edges. They show that $p = \frac{\log n}{n}$, $p = \frac{2 \log n}{n}$, $p = \frac{3 \log n}{n}$, and $p = \frac{4 \log n}{n}$ are sharp thresholds to guarantee that the resulting temporal graph G satisfies the following respective conditions asymptotically almost surely: a fixed pair of vertices can reach each other via temporal paths in G , there is some vertex s which can reach all other vertices in G via temporal paths, G is temporally connected, G and admits a temporal spanner with $2n - 4$ edges (which is tight when time-labels are locally distinct).

Besides temporal graphs, other models to represent graphs or paths that evolve over time have been considered in the literature, we refer the interested reader to [9] for a survey.

Finally, as we already mentioned, there is a large body of literature concerning spanners on static graphs (see [1] for a survey on the topic) and clustering techniques similar to the ones we employ on temporal cliques have proven to be a useful tool to design sparse spanner also in this setting (see, e.g., [5, 6]).

A reader that is already familiar with the area might notice that our upper bound of $\tilde{O}(n^{1+\frac{1}{k}})$ on the size of a temporal $(2k-1)$ -spanner of a *temporal clique*, happens to resemble the classical upper bound of $O(n^{1+\frac{1}{k}})$ on the size of a $(2k-1)$ -spanner of a *general static graph* [3]. Nevertheless, the first result only applies to complete (temporal) graphs and requires different technical arguments to handle temporal paths.

2 Model and preliminaries

Let $G = (V, E)$ be an undirected *temporal graph* with n vertices, and a labeling function $\lambda : E \rightarrow \mathbb{N}^+$ that assigns a *time-label* $\lambda(e)$ to each edge e . If G is complete we will say that it is a *temporal clique*. A temporal path π from vertex u to vertex v is a path in G from u to v such that the sequence e_1, e_2, \dots, e_k of edges traversed by π satisfies $\lambda(e_i) \leq \lambda(e_{i+1})$ for all $i = 1, \dots, k-1$. We denote with $|\pi|$ the length of π , i.e., the number of its edges. A shortest temporal path from vertex u to vertex v is a temporal path from u to v with minimum length. We denote with $d_G(u, v)$, the length of a shortest temporal path from u to v in G . Given a generic graph H , we denote by $V(H)$ its vertex-set and by $E(H)$ its edge-set.

For $\alpha \geq 1$ and $\beta \geq 0$, a temporal (α, β) -spanner of G is a (temporal) subgraph H of G such that $V(H) = V$ and $d_H(u, v) \leq \alpha \cdot d_G(u, v) + \beta$, for each $u, v \in V$. We call a temporal (α, β) -spanner: (i) temporal α -spanner if $\beta = 0$, (ii) temporal β -additive spanner if $\alpha = 1$, (iii) temporal preserver if $\alpha = 1$ and $\beta = 0$. We say that H is a *single-source* temporal (α, β) -spanner w.r.t. a vertex $s \in V$, if $d_H(s, v) \leq \alpha \cdot d_G(s, v) + \beta$, for each $v \in V$. The *size* of a temporal spanner is the number of its edges.

We define the *lifetime* L of G as the number of distinct time-labels of its edges. Furthermore, we assume w.l.o.g. that each time instant in $\{1, \dots, L\}$ is used by at least one time-label (since otherwise we can replace each time-label with its rank in the set $\{\lambda(e) \mid e \in E\}$), so that $L = \max_{e \in E} \lambda(e)$.

We will make use of the following well-known result:

► **Lemma 1.** *Given a collection \mathcal{S} of subsets of $\{1, \dots, n\}$, where each subset has size at least ℓ and $|\mathcal{S}|$ is polynomially bounded in n , we can find in polynomial time a subset $R \subseteq \{1, \dots, n\}$ of size $O((n/\ell) \log n)$ that hits all subsets in the collection, i.e., $R \cap S \neq \emptyset$ for all $S \in \mathcal{S}$.*

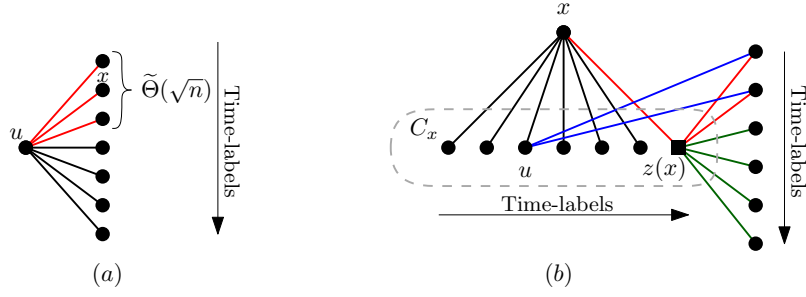
3 Spanners for temporal cliques

In this section, we design an algorithm such that, given a temporal clique G , returns a temporal $(2k-1)$ -spanner H of G with size $\tilde{O}(n^{1+\frac{1}{k}})$, for any integer $k > 1$. We also provide a temporal clique G for which any temporal 2-spanner of G has size $\Omega(n^2)$.

Before describing the algorithm for constructing temporal $(2k-1)$ -spanners, we show as a warm up how to construct a temporal 3-spanner and a temporal 5-spanner of size $\tilde{O}(n^{1+\frac{1}{2}})$ and $\tilde{O}(n^{1+\frac{1}{3}})$, respectively.

3.1 Our temporal 3-spanner

Given a temporal clique G , we construct a temporal 3-spanner H of G via a clustering technique. For each $u \in V$, we select a set E_u containing the $\Theta(\sqrt{n \log n})$ edges incident to u having the smallest labels (ties are broken arbitrarily). We define $S_u = \{v \in V \mid (u, v) \in E_u\}$. Next, we find a hitting set $R \subseteq V$ of the collection $\{S_u\}_{u \in V}$. Thanks to Lemma 1, we can deterministically compute a hitting set of size $|R| = O(\sqrt{n \log n})$.



■ **Figure 1** (a) A vertex u and its neighbours in G , the edges are sorted top-down by increasing time-label. The red edges are those belonging to E_u while the black edges belong to $E(G) \setminus E_u$. (b) An example of a cluster C_x where $x \in R$. The edges incident to x are sorted from left to right by increasing-time label. The black and red edges are added to $E(H)$ during the initialization phase, in particular the red edges are those belonging to $E_{z(x)}$. The blue edges are added w.r.t. u to $E(H)$ during the first augmentation. The green edges belong to $E(G) \setminus E_{z(x)}$ and are added to $E(H)$ during the second augmentation.

We partition the vertices of V into $|R|$ clusters. More precisely, we create a cluster $C_x \subseteq V$ for each vertex $x \in R$. Each vertex $u \in V$ belongs to exactly one arbitrarily chosen cluster C_x that satisfies $x \in S_u$, i.e., x hits S_u . We call x the center of cluster C_x .⁴ Moreover, we choose the special vertex of cluster C_x as a vertex $z(x)$ in C_x that maximizes the label of the edge $(x, z(x))$.

Notice that, for every $x \in R$ and $u \in C_x$, u can reach $z(x)$ via a temporal path of length at most 2 in G by using the edges (u, x) and $(x, z(x))$ since, by definition of $z(x)$ and S_u , we have $\lambda(u, x) \leq \lambda(x, z(x))$.

We now build our temporal spanner H of G . The set of edges $E(H)$ is constructed in three phases (See Figure 1 for an example of the whole construction):

Initialization: For each $u \in V$, we add the edges in E_u to $E(H)$;

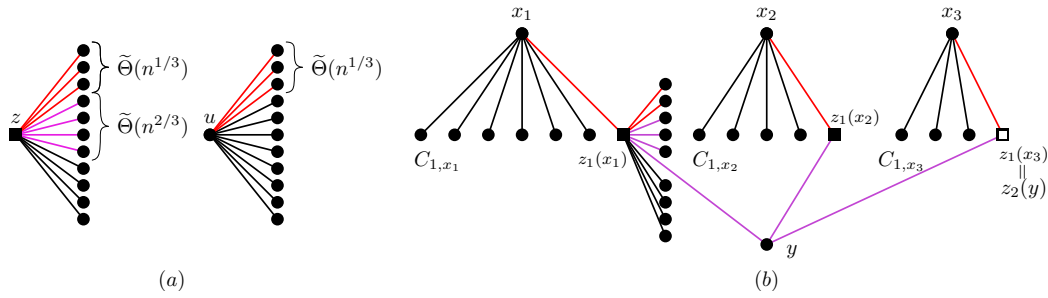
First Augmentation: For every $u \in V$, we add the edges in $E_{u, z(x)} = \{u\} \times S_{z(x)}$ to $E(H)$, where x is the center of the cluster containing u ;

Second Augmentation: For each $x \in R$, we add the edges in $E_{z(x), V} = \{z(x)\} \times V$ to $E(H)$. It is easy to see that H contains $O(n\sqrt{n \log n})$ edges. We now show that for any $u, v \in V$ there is a temporal path from u to v of length at most 3 in H . Indeed, let $x \in R$ be the center of the cluster C_x containing u . If $v = z(x)$ then, since $u \in C_x$, the initialization phase ensures that $(u, x) \in E(H)$ and $(x, z(x)) \in E(H)$, which form a temporal path as we already discussed above. We hence assume that $v \neq z(x)$. If $(z(x), v) \in E_{z(x)}$ then the first augmentation phase added $(u, v) \in E_{u, z(x)}$ to $E(H)$, which is a temporal path of length one from u to v . Otherwise $(z(x), v) \in E(G) \setminus E_{z(x)}$ and, the second augmentation phase added edge $(z(x), v)$ to $E(H)$. Moreover, since $(z(x), v) \notin E_{z(x)}$, $(z(x), v)$ is not among the $\Theta(\sqrt{n \log n})$ edges incident to $z(x)$ with lowest labels. As a consequence, since $(x, z(x)) \in E_{z(x)}$, we have $\lambda(x, z(x)) \leq \lambda(z(x), v)$. Hence, the edges (u, x) , $(x, z(x))$, and $(z(x), v)$ form a temporal path of length 3 from u to v in H .

3.2 Our temporal 5-spanner

We show how to modify the construction of a temporal 3-spanner given in previous section in order to obtain a temporal 5-spanner of size $\tilde{O}(n^{4/3})$. The idea is to replace the single-level clustering of Section 3.1 with a two-level clustering, where the second-level clustering

⁴ Here and throughout the paper, the center of a cluster is not required to belong to the cluster itself.



■ **Figure 2** (a) Two vertices u and z of G , where $z \in Z_1$ and the red edges belong to $E_{1,u}$ and $E_{1,z}$, respectively. For vertex z the purple edges belong to $E_{2,z}$. (b) A two level clustering. The level one consists of three cluster C_{1,x_1} , C_{1,x_2} , C_{2,x_3} . The level-two cluster $C_{2,y}$, with $y \in R_2$, contains vertices $z_1(x_1)$, $z_1(x_2)$ and $z_1(x_3)$, where $z_2(y) = z_1(x_3)$.

partitions the special vertices of the first level clustering and the number of selected clusters decreases as we move from the first level to the second one.

The level-one clustering is built similarly to the one used in our temporal 3-spanner. For each vertex $u \in V$ we define sets $E_{1,u}$ and $S_{1,u}$ where $E_{1,u}$ consists of the $\Theta(n^{1/3} \log^{2/3} n)$ edges with the smallest label among those incident to u (ties are broken arbitrarily) and $S_{1,u} = \{v \in V \mid (u, v) \in E_{1,u}\}$. We compute a hitting set R_1 of the collection $\{S_{1,u}\}_{u \in V}$, where R_1 has size $O(n^{2/3} \log^{1/3} n)$ thanks to Lemma 1. We partition the vertices of V into $|R_1|$ clusters $C_{1,x}$, for each $x \in R_1$, as before, and let $z_1(x)$ the vertex in $C_{1,x}$ that maximizes the label of the edge $(x, z_1(x))$.

The level-two clustering is built on top of the vertices $Z_1 = \{z_1(x) \mid x \in R_1\}$. For each $u \in Z_1$, we define $E_{2,u}$ as a set of $\Theta(n^{2/3} \log^{1/3} n)$ edges with the smallest label among those that are incident to u but do not belong to $E_{1,u}$. We also define a corresponding set $S_{2,u} = \{v \in V \mid (u, v) \in E_{2,u}\}$. We once again invoke Lemma 1 to compute a hitting set R_2 of size $O(n^{1/3} \log^{2/3} n)$ of the collection $\{S_{2,u}\}_{u \in Z_1}$. Based on R_2 , we partition the special vertices in Z_1 by associating each $u \in Z_1$ to an arbitrary cluster $C_{2,y}$ centered in $y \in R_2$ such that $y \in S_{2,u}$. Each cluster $C_{2,y}$ has an associated special vertex $z_2(y) \in C_{2,y}$ chosen among the ones that maximize the label of the edge $(y, z_2(y))$, see Figure 2.

We are now ready to build our temporal 5-spanner H . As before, the set of edges $E(H)$ is constructed in three phases:

Initialization: For each $u \in V$, we add the edges in $E_{1,u}$ to $E(H)$ and, for each $u \in Z_1$, we add the edges in $E_{2,u}$ to $E(H)$;

First Augmentation: For every $u \in V$, we add the edges in $E_{u,z_1(x)} = \{u\} \times S_{1,z_1(x)}$ to $E(H)$, where x is the center of the cluster containing u . Moreover, for each $z \in Z_1$ we add the edges in $\{z\} \times (S_{1,z_2(y)} \cup S_{2,z_2(y)})$ to $E(H)$, where y is the center of the level-two cluster $C_{2,y}$ containing z ;

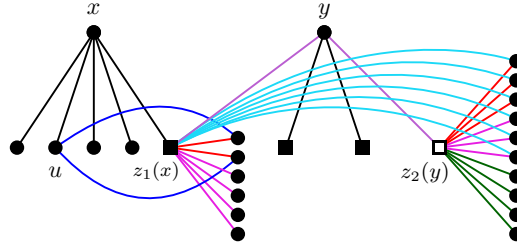
Second Augmentation: For each $y \in R_2$, we add the set $\{z_2(y)\} \times V$ to $E(H)$.

See Figure 3 for an example of the whole construction. We now show that H is a 5-spanner of size $O(n^{4/3} \log^{2/3} n)$.

► **Lemma 2.** *Let $u, v \in V$. There is a temporal path from u to v of length at most 5 in H .*

Proof. Let $x \in R_1$ be the center of the level-one cluster $C_{1,x}$ containing u and $y \in R_2$ be the center of the level-two cluster $C_{2,y}$ containing $z_1(x)$.

We first show that in H there exists a temporal path π of length 4 from u to $z_2(y)$ consisting of the sequence of edges (u, x) , $(x, z_1(x))$, $(z_1(x), y)$, $(y, z_2(y))$. Notice that, the edges (u, x) , $(x, z_1(x))$, $(z_1(x), y)$, $(y, z_2(y))$ belong to $E_{1,u}$, $E_{1,z_1(x)}$, $E_{2,z_1(x)}$, and $E_{2,z_2(y)}$,



■ **Figure 3** An example of a two-level cluster and of the edges added to $E(H)$ during the spanner construction. The black, red and purple edges are added during initialization phase. In particular, for every $u \in V$, the red edges are those in $E_{1,u}$ and, for every $z \in Z_1$, the purple edges are those in $E_{2,z}$. The dark blue and light blue edges are those added to $E(H)$ during the first augmentation phase. The green edges are the edges added to $E(H)$ during the second augmentation phase.

respectively. Moreover, the initialization phase ensures that they all belong to $E(H)$. Then, by definition of $z_1(x)$, we have $\lambda(u, x) \leq \lambda(x, z_1(x))$. Moreover, since $(x, z_1(x)) \in E_{1,z_1(x)}$ and $(z_1(x), y) \in E_{2,z_1(x)}$, then $\lambda(x, z_1(x)) \leq \lambda(z_1(x), y)$. Finally, $(y, z_2(y)) \in E_{2,z_2(y)}$ and, by definition of $z_2(y)$, we have $\lambda(z_1(x), y) \leq \lambda(y, z_2(y))$.

If $v = z_2(y)$, then u can reach v via a temporal path of length 4 in H , by using π . Moreover, if $v = z_1(x)$ then u can reach v via a temporal path of length 2 by using the subpath π_1 of π consisting of the edges (u, x) and $(x, z_1(x))$. Otherwise, we can build a temporal path to v by considering one of following three cases (to be checked in order):

- If $(z_1(x), v) \in E_{1,z_1(x)}$, then $v \in S_{1,z_1(x)}$ and, due to the first augmentation phase, we have that $(u, v) \in E(H)$.
- If $(z_2(y), v) \in E_{1,z_2(y)} \cup E_{2,z_2(y)}$, then vertex $v \in S_{1,z_2(y)} \cup S_{2,z_2(y)}$ and the first augmentation phase ensures that $(z_1(x), v) \in E(H)$. Moreover, since $(z_1(x), v) \notin E_{1,z_1(x)}$, we have that $\lambda(x, z_1(x)) \leq \lambda(z_1(x), v)$. Hence the concatenation of π_1 with the edge $(z_1(x), v)$ yields a temporal path of length 3 from u to v in H .
- If $(z_2(y), v) \notin (E_{1,z_2(y)} \cup E_{2,z_2(y)})$, the second augmentation phase ensures that $(z_2(y), v) \in E(H)$. Moreover, $\lambda(y, z_2(y)) \leq \lambda(z_2(y), v)$. Therefore the concatenation of π with the edge $(z_2(y), v)$ yields a temporal path of length 5 from u to v in H . ◀

▶ **Lemma 3.** *The size of H is $O(n^{4/3} \log^{2/3} n)$.*

3.3 Our temporal $(2k - 1)$ -spanner

In this section, we describe an algorithm that, given an integer $k \geq 2$ and a temporal clique G of n vertices, returns a temporal $(2k - 1)$ -spanner of G with size $O(k \cdot n^{1+\frac{1}{k}} \log^{\frac{k-1}{k}} n)$.

The idea is to define a hierarchical clustering of G , where a generic level- i clustering partitions the special vertices of the level- $(i - 1)$ clustering and determines the special vertices of level i . As we move from one clustering level to the next, the number of clusters decreases by a factor of roughly $n^{\frac{1}{k}}$, thus allowing us to add an increasing number of edges incident to the special vertices into the spanner.

We ensure that each vertex $u \in V$ can reach some special vertex by moving upwards in the clustering hierarchy. These special vertices work as hubs, i.e., each of them allows to directly reach a subset of vertices of V , and some special vertex of higher level (via a temporal path of length at most 2). Then u can reach any vertex in $v \in V$ by first reaching a suitable special vertex z in the hierarchy, and then following the edge (z, v) .

■ **Algorithm 1** Computes a temporal $(2k - 1)$ -spanner.

```

Input : A temporal clique  $G$ ;
Output : A temporal  $(2k - 1)$ -spanner of  $G$ ;

1  $Z_0 \leftarrow V$ ;
2 foreach  $u \in V$  do  $E(u) \leftarrow \{(u, v) \mid v \in V\}$ ;
3 for  $i = 1, \dots, k - 1$  do
4   foreach  $u \in Z_{i-1}$  do
5      $E_{i,u} \leftarrow$  set of the first min-time label  $n^{\frac{i}{k}} \log n$  edges of  $E(u)$ ;
6      $E(u) \leftarrow E(u) \setminus E_{i,u}$ ;
7      $S_{i,u} \leftarrow \{v \in V : (u, v) \in E_{i,u}\}$ ;
8    $R_i \leftarrow$  hitting set of  $\{S_{i,u}\}_{u \in Z_{i-1}}$  computed as in Lemma 1;
9    $C \leftarrow \emptyset$ ; // Set of vertices in  $Z_{i-1}$  that are already clustered
10  foreach  $x \in R_i$  do
11     $C_{i,x} = \{u \in Z_i \setminus C : x \in S_{i,u}\}$ ;
12     $z_i(x) \leftarrow \arg \max_{u \in C_{i,x}} \{\lambda(u, x)\}$ ;
13     $C \leftarrow C \cup C_{i,x}$ ;
14   $Z_i \leftarrow \{z_i(x) \in Z_{i-1} : x \in R_i\}$ ;
15  $H \leftarrow (V, \emptyset)$  for  $i = 1$  to  $k - 1$  do // Initialization
16   foreach  $u \in Z_{i-1}$  do  $E(H) \leftarrow E(H) \cup E_{i,u}$ ;
17 for  $i = 1, \dots, k - 1$  do // First augmentation
18   foreach  $u \in Z_{i-1}$  do
19     Let  $x \in R_i$  such that  $u \in C_{i,x}$ ;
20      $E(H) \leftarrow E(H) \cup \{u\} \times \{\bigcup_{j=1}^i S_{z_i(x),j}\}$ ;
21 foreach  $z \in Z_{k-1}$  do  $E(H) \leftarrow E(H) \cup \{z\} \times V$ ; // Second Augmentation
22 return  $H$ ;

```

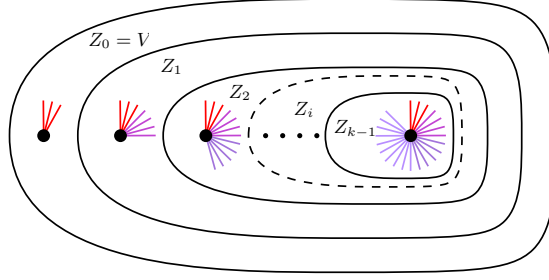
We build our clustering in $k - 1$ rounds indexed from 1 to $k - 1$ (a detailed pseudocode is given in Algorithm 1), where the generic i -th round defines a set Z_i of level- i special vertices. Initially, $Z_0 = V$, i.e., all vertices are special vertices of level 0. During the i -th round, the level- i clustering is computed from the set of vertices in Z_{i-1} defined at the previous round as follows.

For each $u \in Z_{i-1}$, we let $E_{i,u}$ be a set of $\delta_i = \Theta(n^{\frac{i}{k}} \log^{\frac{k-i}{k}} n)$ edges with the smallest label among those that are incident to u but do not belong to $\bigcup_{j=1}^{i-1} E_{u,j}$, and we denote by $S_{i,u} = \{v \in V : (u, v) \in E_{i,u}\}$ the set containing the endvertices of the edges incident to u in $E_{i,u}$. We now compute a hitting set $R_i \subseteq V$ of the collection $\{S_{i,u} \mid u \in Z_{i-1}\}$ having size at most $O(\frac{n}{\delta_i} \log n)$. Lemma 1 guarantees that R_i always exists. Notice that, as i increases, the time labels of the edges in $E_{i,u}$ became larger, δ_i increases, and $|R_i|$ decreases.

We now partition the vertices in Z_{i-1} into $|R_i|$ clusters $C_{i,x}$, one for each $x \in R_i$. We do so by adding each vertex $u \in Z_{i-1}$ into an arbitrary cluster $C_{i,x}$ such that $x \in S_{i,u}$. We call x the center of the cluster $C_{i,x}$. Moreover, for each cluster $C_{i,x}$, we choose a special vertex $z_i(x) \in C_{i,x}$ as a vertex that maximizes the label of edge $(x, z_i(x))$.

Once the hierarchical clustering is built, our algorithm proceeds to construct a temporal $(2k - 1)$ -spanner H of G . At the beginning $H = (V, \emptyset)$, then edges are added to H in the following three phases:

19:10 Sparse Temporal Spanners with Low Stretch



■ **Figure 4** The set of edges selected for each vertex during the initialization phase.

Initialization: For each $u \in V$, we add to $E(H)$ all the edges in the sets $E_{i,u}$ for $i = 1, \dots, j+1$, where j is the largest integer between 0 and $k-2$ for which $u \in Z_j$, see Figure 4.

First Augmentation: For each $i = 1, \dots, k-1$ and each $u \in Z_{i-1}$, we consider the center $x \in R_i$ of the level- i cluster $C_{i,x}$ containing u , and we add to $E(H)$ all the edges (u, v) with $v \in \bigcup_{j=1}^i S_{j,z_i(x)}$.

Second Augmentation: We add to $E(H)$ all edges incident to some vertex in Z_{k-1} .

We now show that all vertices are at distance at most $2k-1$ in H , and that the size of H is $O(k \cdot n^{1+\frac{1}{k}} \log^{\frac{k-1}{k}} n)$.

► **Lemma 4.** *For every $u, v \in V(G)$, $d_H(u, v) \leq (2k-1)d_G(u, v)$.*

Proof. Let $z_0 = u$ and, for $i = 1, \dots, k-1$, let $z_i = z_i(x_i)$ where $x_i \in R_i$ is the center of the cluster C_{i,x_i} containing z_{i-1} . The initialization phase ensures that, for any i , there exists a temporal path from z_0 to z_i in H of length $2i$ entering z_i with the edge $(x_i, z_i) \in E_{i,z_i}$.⁵ Indeed, π_i can be chosen as the path that traverses edge $(z_{i-1}, x_i) \in E_{i,z_{i-1}}$ and edge $(x_i, z_i) \in E_{i,z_i}$, in this order. Notice that, by definition of z_i , $\lambda(z_{i-1}, x_i) \leq \lambda(x_i, z_i)$. Moreover, if $i < k-1$, $\lambda(x_i, z_i) \leq \lambda(z_i, x_{i+1})$ since $(x_i, z_i) \in E_{i,z_i}$ while $(z_i, x_{i+1}) \in E_{i+1,z_i}$. See Figure 5.

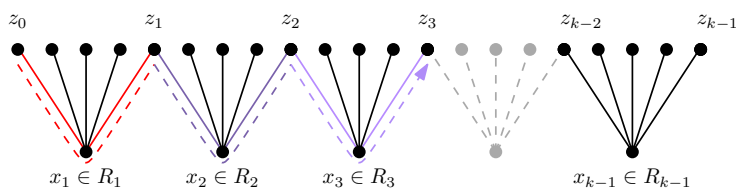
If $v = z_i$ for some $i = 1, \dots, k-1$ then, from the discussion above, we know that π_i is a temporal path from u to v in H of length $2i < 2k-1$. Otherwise, we distinguish two cases depending on whether there exists some $i = 1, \dots, k-1$ such that $(z_i, v) \in \bigcup_{j=1}^i E_{j,z_i}$.

Suppose that the above condition is met, and let $i > 0$ be the minimum index for which $(z_i, v) \in \bigcup_{j=1}^i E_{j,z_i}$. If $\lambda(z_i, v) \geq \lambda(x_i, z_i)$, then π_i followed by edge (z_i, v) , is a temporal path from u to v of length $2i+1 \leq 2k-1$. If $\lambda(z_i, v) < \lambda(x_i, z_i)$ then, since $(z_i, v) \in \bigcup_{j=1}^i E_{j,z_i}$, we have $v \in \bigcup_{j=1}^i S_{j,z_i}$ and the first augmentation phase adds (z_{i-1}, v) to $E(H)$. By hypothesis we have $(z_{i-1}, v) \notin \bigcup_{j=1}^{i-1} E_{j,z_{i-1}}$ and hence $\lambda(z_{i-1}, v) \geq \lambda(x_{i-1}, z_{i-1})$. This shows that π_{i-1} followed by (z_{i-1}, v) is a temporal path from u to v in H of length $2i-1 \leq 2k-1$.

It only remains to handle the case in which, for every i , we have $(z_i, v) \notin \bigcup_{j=1}^i E_{j,z_i}$. In this case, the algorithm adds (z_{k-1}, v) to $E(H)$ during the second augmentation phase. Moreover, since $\lambda(z_{k-1}, v) \geq \lambda(x_{k-1}, z_{k-1})$, the path π_{k-1} followed by edge (z_{k-1}, v) is a temporal path from u to v in H of length $2k-1$. ◀

► **Theorem 5.** *Given a temporal clique G , for any $k \geq 1$, the above algorithm computes a temporal $(2k-1)$ -spanner H of size $O(k \cdot n^{1+\frac{1}{k}} \log^{\frac{k-1}{k}} n)$.*

⁵ This path is not necessarily a simple path (e.g., when $z_i = z_{i+1}$). The existence of a non-simple temporal path of length ℓ implies the existence of simple temporal path of length at most ℓ .



■ **Figure 5** The hierarchy of clusters for vertex z_0 , where $z_i = z_i(x_i)$ where $x_i \in R_i$ is such that $z_{i-1} \in C_{i,x_i}$. The dashed line is the temporal path π_3 that goes from z_0 to z_3 .

We conclude this section with a simple lower bound on the size of any temporal 2-spanners of a temporal clique.

► **Theorem 6.** *There exists a temporal clique G of n vertices such that any temporal 2-spanner of G has size $\Omega(n^2)$.*

4 Single-source spanners for general temporal graphs

In the first part of this section we design an algorithm that, for every $0 < \varepsilon < n$, builds a single-source temporal $(1 + \varepsilon)$ -spanner of G w.r.t. s of size $O\left(\frac{n \log^4 n}{\log(1+\varepsilon)}\right)$. We observe that, for constant values of ε , the size of the computed spanner is almost linear, i.e., linear up to polylogarithmic factors. The algorithm can be extended so as, for every $1 \leq \beta < n$, it builds a single-source temporal β -additive spanner of G w.r.t. s of size $O\left(\frac{n^2 \log^4 n}{\beta}\right)$.⁶

Our upper bounds leave open the problem of deciding whether a temporal graph G admits a single-source temporal preserver w.r.t. s of size $\tilde{O}(n)$. We answer to this question negatively in the second part of this section. More precisely, we show a temporal graph G of size $\Theta(n^2)$ and a source vertex s for which no edge can be removed if we want to keep a shortest temporal path from s to every other vertex u . The construction can be extended to show a lower bound of $\Omega(n^2/\beta)$ on the size of single-source temporal β -additive spanners, for every $\beta \geq 1$. This implies that our upper bound on the size of single-source temporal additive spanners is asymptotically optimal, up to polylogarithmic factors.

4.1 Our upper bound

In this section we present an algorithm that, for every $0 < \varepsilon < n$, computes a single-source temporal $(1 + \varepsilon)$ -spanner of G w.r.t. s of size $O\left(\frac{n \log^4 n}{\log(1+\varepsilon)}\right)$ in polynomial time.⁷

In the following we say that a temporal path is τ -restricted if it uses edges of time-label of at most τ . Our algorithm computes a spanner that, for every $\tau = 1, \dots, L$, contains $(1 + \varepsilon)$ -approximate τ -restricted temporal paths from s to any vertex v (recall that L is the lifetime of G). More formally, for two vertices u and v of G , we denote by $d_G^{\leq \tau}(u, v)$ the length of a shortest τ -restricted temporal path from u to v in G . We assume $d_G^{\leq \tau}(u, v) = +\infty$ when G does not contain a τ -restricted temporal path from u to v . The single-source temporal $(1 + \varepsilon)$ -spanner H of G w.r.t. s computed by our algorithm is such that, for every $v \in V$, and for every $\tau = 1, \dots, L$, $d_H^{\leq \tau}(s, v) \leq (1 + \varepsilon)d_G^{\leq \tau}(s, v)$.

⁶ We refer the interested reader to the full version of this paper for details.

⁷ Our algorithm also works in the case of directed temporal graphs and/or multiple time-labels. Both the algorithm and the stretch analysis require no modification. Regarding the running time, we only need to observe that τ -restricted shortest paths can be computed in polynomial time even in directed/multiple-label temporal graphs.

19:12 Sparse Temporal Spanners with Low Stretch

■ **Algorithm 2** Computes a set Π_v of temporal paths from s to v in G that provides a good approximation of any shortest τ -restricted temporal path from s to v in G .

Input : A temporal graph G (with lifetime L), a source vertex $s \in V$, and a vertex $v \in V$.
Output : A set Π_v of temporal paths from s to v in G such that, for every $\tau = 1, \dots, L$, there exists a τ -restricted temporal path $\pi \in \Pi_v$ such that $|\pi| \leq (1 + \delta)d_G^{\leq \tau}(s, v)$.

```

1  $\Pi_v \leftarrow \emptyset$ ;  $t \leftarrow +\infty$ ;
2 for  $\tau = 1$  to  $L$  do
3   if  $d_G^{\leq \tau}(s, v) \neq +\infty$  and  $d_G^{\leq \tau}(s, v) < \frac{t}{1+\delta}$  then
4     Let  $\pi$  be a shortest  $\tau$ -restricted temporal path from  $s$  to  $v$  in  $G$ ;
5      $\Pi_v \leftarrow \Pi_v \cup \{\pi\}$ ;
6      $t \leftarrow |\pi|$ ;
7 return  $\Pi_v$ ;
```

For technical convenience, in the following we design an algorithm that, for any $0 < \delta < n$ and any positive integer k , builds a single-source temporal $(1 + \delta)^k$ -spanner of G w.r.t. s of size $O\left(\frac{kn^{1+\frac{1}{k}} \log^{2-\frac{1}{k}} n}{\log(1+\delta)}\right)$. The desired bound of $O\left(\frac{n \log^4 n}{\log(1+\varepsilon)}\right)$ on the size of the single-source temporal $(1 + \varepsilon)$ -spanner is obtained by choosing $k = \lceil \log n \rceil$ and $\delta = (1 + \varepsilon)^{\frac{1}{\log n}} - 1$.

Our algorithm uses a subroutine that, for a given vertex v of G , computes a set Π_v of $O\left(\frac{\log n}{\log(1+\delta)}\right)$ temporal paths from s to v of G such that, for every $\tau = 1, \dots, L$, Π_v contains a τ -restricted temporal path π satisfying $|\pi| \leq (1 + \delta)d_G^{\leq \tau}(s, v)$.

The subroutine (see Algorithm 2 for the pseudocode) builds Π_v iteratively by adding a subset of shortest τ -restricted temporal paths from s to v in G , where $\tau = 1, \dots, L$. We do so by scanning shortest τ -restricted temporal paths from s to v in increasing order of values of τ . The scanned path π is added to Π_v if no other path already contained in Π_v has a length of at most $(1 + \delta)|\pi|$. The next lemma shows the correctness of our subroutine and bounds the number of paths contained in Π_v .

► **Lemma 7.** *For every $\tau = 1, \dots, L$, there is a τ -restricted temporal path π in Π_v such that $|\pi| \leq (1 + \delta)d_G^{\leq \tau}(s, v)$. Moreover, $|\Pi_v| = O\left(\frac{\log n}{\log(1+\delta)}\right)$.*

In the rest of this section, for any given temporal path π , we denote by $\pi(\ell)$ the subpath of π containing the last $\min\{\ell, |\pi|\}$ edges of π . We observe that $\pi(\ell) = \pi$ when $|\pi| \leq \ell$. Moreover, for two vertices u and v of a temporal path π that visits u before v , we denote by $\pi[u, v]$ the temporal subpath of π from u to v .

Before diving into the technical details, we describe the main idea of our algorithm and show how we can use it to build a single-source temporal $(1 + \delta)^2$ -spanner of G w.r.t. s of size $O\left(\frac{n^{3/2} \log^{3/2} n}{\log(1+\delta)}\right)$.

For technical convenience, let $R_0 = V$ and $\mathcal{P}_0 = \bigcup_{v \in R_0} \Pi_v$. In principle, we could build our single-source temporal $(1 + \delta)$ -spanner of G w.r.t. s by simply setting its edge set to $\bigcup_{\pi \in \mathcal{P}_0} E(\pi)$. Unfortunately, Lemma 7 alone is insufficient to provide a subquadratic upper bound on the size of this spanner. Therefore, to obtain a spanner of truly subquadratic size, we compute a single-source temporal $(1 + \delta)^2$ -spanner H of G w.r.t. s instead.

We build H by adding all the *short* temporal paths in \mathcal{P}_0 , i.e., all paths with at most ℓ_0 edges for a suitable choice of ℓ_0 , and by replacing each *long* temporal path $\pi \in \mathcal{P}_0$ from s to some vertex v with the shortest temporal path from s to x in Π_x , for some vertex x that *hits* $\pi(\ell_0)$, combined with $\pi[x, v]$.

■ **Algorithm 3** Single-source temporal spanner of a temporal graph G .

Input : A temporal graph $G = (V, E)$ of n vertices and a source vertex $s \in V$.
Output : A single-source temporal spanner H of G w.r.t. s .

- 1 **for** $i = 0, \dots, k-1$ **do** $\ell_i \leftarrow n^{\frac{i+1}{k}} \log^{1-\frac{i+1}{k}} n$;
- 2 **foreach** $v \in V$ **do** use Algorithm 2 to compute Π_v ;
- 3 $R_0 \leftarrow V$; $\mathcal{P}_0 \leftarrow \{\pi \in \Pi_v \mid v \in R_0\}$;
- 4 **for** $i = 1, \dots, k-1$ **do**
- 5 $\mathcal{P}_{i-1}^{long} = \{\pi \in \mathcal{P}_{i-1} \mid |\pi| > \ell_i\}$;
- 6 $R_i \leftarrow$ hitting set of $\{\pi(\ell_{i-1}) \mid \pi \in \mathcal{P}_{i-1}^{long}\}$ computed as in Lemma 1;
- 7 $\mathcal{P}_i \leftarrow \bigcup_{v \in R_i} \Pi_v$;
- 8 **return** $H = \left(V, \bigcup_{i=0}^{k-1} \bigcup_{\pi \in \mathcal{P}_i} E(\pi(\ell_i)) \right)$;

In more details, we define $\ell_0 = \sqrt{n \log n}$ and we introduce a new parameter $\ell_1 = n$. We say that a temporal path $\pi \in \mathcal{P}_0$ is *short* if $|\pi| \leq \ell_0$; it is *long* otherwise. Let $\mathcal{P}_0^{long} = \{\pi \in \mathcal{P}_0 \mid |\pi| > \ell_0\}$ be the subset of long temporal paths in \mathcal{P}_0 . We compute a set R_1 that hits $\{\pi(\ell_0) \mid \pi \in \mathcal{P}_0^{long}\}$ using Lemma 1, and we then use this set to define a new collection of temporal paths $\mathcal{P}_1 = \bigcup_{v \in R_1} \Pi_v$.⁸ The edge set of H is defined as $E(H) = \bigcup_{i \in \{0,1\}} \bigcup_{\pi \in \mathcal{P}_i} E(\pi(\ell_i))$. The next lemma shows that this simple algorithm already computes a single-source temporal $(1 + \delta)^2$ -spanner of G w.r.t. s of truly subquadratic size.

► **Lemma 8.** *For every $\tau = 1, \dots, L$ and every $v \in V$, $d_H^{\leq \tau}(s, v) \leq (1 + \delta)^2 d_G^{\leq \tau}(s, v)$. Moreover, the size of H is $O\left(\frac{n^{3/2} \log^{3/2} n}{\log(1+\delta)}\right)$.*

The technique we used to replace each of the temporal paths in \mathcal{P}_0^{long} with a temporal path that is longer by a factor of at most $(1 + \delta)$ can be applied recursively on the set \mathcal{P}_1 , for a suitable choice of ℓ_1 , to obtain an even sparser spanner. As we show now, $k-1$ levels of recursion allow us to compute a single-source temporal $(1 + \delta)^k$ -spanner H of G w.r.t. s of size $O\left(\frac{kn^{1+\frac{1}{k}} \log^{2-\frac{1}{k}} n}{\log(1+\delta)}\right)$.

In the following we provide the technical details (see Algorithm 3 for the pseudocode). For every $i = 0, \dots, k-1$, let $\ell_i = n^{\frac{i+1}{k}} \log^{1-\frac{i+1}{k}} n$. As before, let $R_0 = V$ and $\mathcal{P}_0 = \bigcup_{v \in R_0} \Pi_v$. During the i -th iteration, the algorithm computes a set R_i that hits $\{\pi(\ell_{i-1}) \mid \pi \in \mathcal{P}_{i-1}^{long}\}$, where $\mathcal{P}_{i-1}^{long} = \{\pi \in \mathcal{P}_{i-1} \mid |\pi| > \ell_{i-1}\}$ is the set of *long* temporal paths of \mathcal{P}_{i-1} . The i -th iteration ends by computing the set $\mathcal{P}_i = \bigcup_{v \in R_i} \Pi_v$ that is used in the next iteration. The edge set of the graph H that is returned by the algorithm is $E(H) := \bigcup_{i=0}^{k-1} \bigcup_{\pi \in \mathcal{P}_i} E(\pi(\ell_i))$.

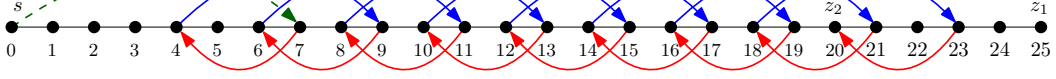
► **Theorem 9.** *For every $\tau = 1, \dots, L$, for every $i = 1, \dots, k$, and for every $v \in R_{k-i}$, we have that $d_H^{\leq \tau}(s, v) \leq (1 + \delta)^i d_G^{\leq \tau}(s, v)$. Moreover, the size of H is $O\left(\frac{kn^{1+\frac{1}{k}} \log^{2-\frac{1}{k}} n}{\log(1+\delta)}\right)$.*

Proof. We start proving the first part of the theorem statement. The proof is by induction on i . Fix a vertex $v \in R_{k-i}$ such that $d_G^{\leq \tau}(s, v)$ is finite.

For the base case $i = 1$, we observe that Π_v is entirely contained in H by construction. Therefore, by Lemma 7, $d_H^{\leq \tau}(s, v) \leq (1 + \delta) d_G^{\leq \tau}(s, v)$ and the claim follows.

We now prove the inductive case. We assume that the claim holds for $i-1$ and we prove it for i . Let $\pi \in \Pi_v$ be a shortest τ -restricted temporal path from s to v among those in Π_v . By Lemma 7, $|\pi| \leq (1 + \delta) d_G^{\leq \tau}(s, v)$. Moreover, by definition, $\pi \in \mathcal{P}_{k-i}$. If π is short, i.e.,

⁸ With a little abuse of notation, R_1 hits a temporal path π if R_1 hits $V(\pi)$.



■ **Figure 6** Example of the lower bound with $h = 2$ and $\beta = 0$. Path π_1 consists of the black edges. Path π_2 is built on top of π_1 , whose vertices have been numbered in order of a traversal from s , and consists of: the first hop from s to μ (in green), the forward hops (in blue), and the backward hops (in red). The arrows on the edges of π_2 are directed away from s along π_2 .

$|\pi| \leq \ell_{k-i}$, then π is entirely contained in H and therefore $d_H^{\leq \tau}(s, v) \leq (1 + \delta)d_G^{\leq \tau}(s, v) \leq (1 + \delta)^{k-i}d_G^{\leq \tau}(s, v)$. So, in the following we assume that π is long. Let $x \in R_{k-i+1}$ be a vertex that hits $\pi(\ell_{k-i})$. By construction, the path $\pi[x, v]$, being a subpath of $\pi(\ell_{k-i})$, is entirely contained in H . Let τ' be the label of the edge incident to x in $\pi[x, v]$. Clearly, $\tau' \leq \tau$. Moreover, by inductive hypothesis, $d_H^{\leq \tau'}(s, x) \leq (1 + \delta)^{i-1} \cdot |\pi[s, x]|$.

As a consequence, $d_H^{\leq \tau}(s, v) \leq d_H^{\leq \tau'}(s, x) + |\pi[x, v]| \leq (1 + \delta)^{i-1} \cdot |\pi[s, x]| + |\pi[x, v]| \leq (1 + \delta)^{i-1} \cdot |\pi| \leq (1 + \delta)^i d_G^{\leq \tau}(s, v)$.

To bound the size of H , we first observe that, for each $v \in V$, $|\Pi_v| = O\left(\frac{\log n}{\log(1+\delta)}\right)$ by Lemma 7. Next, using Lemma 1, we observe that each R_i , with $i \geq 1$, has size $|R_i| = O\left(\frac{n \log n}{\ell_{i-1}}\right) = O\left(n^{1-\frac{i}{k}} \log^{\frac{i}{k}} n\right)$. Furthermore, also $|R_0| = n = n^{1-\frac{0}{k}} \log^{\frac{0}{k}} n$. Therefore, for every $i = 0, \dots, k-1$, we have $|R_i| \ell_i = O\left(n^{1+\frac{1}{k}} \log^{1-\frac{1}{k}} n\right)$. As a consequence,

$$\sum_{\pi \in \mathcal{P}_i} |\pi(\ell_i)| = \sum_{v \in R_i} \sum_{\pi \in \Pi_v} |\pi(\ell_i)| = O\left(|R_i| \ell_i \frac{\log n}{\log(1+\delta)}\right) = O\left(\frac{n^{1+\frac{1}{k}} \log^{2-\frac{1}{k}}}{\log(1+\delta)}\right).$$

Hence, $|E(H)| = \sum_{i=0}^{k-1} \sum_{\pi \in \mathcal{P}_i} |\pi(\ell_i)| = O\left(\frac{kn^{1+\frac{1}{k}} \log^{2+\frac{1}{k}}}{\log(1+\delta)}\right)$. ◀

The following corollary follows by choosing $\tau = L$ and $i = k$ (so that $R_{k-i} = R_0 = V$):

► **Corollary 10.** *Let G be a temporal graph with n vertices and let s be a vertex of G . The graph H returned by Algorithm 3 is a single-source temporal $(1 + \delta)^k$ -spanner of G w.r.t. s of size $O\left(\frac{kn^{1+\frac{1}{k}} \log^{2+\frac{1}{k}}}{\log(1+\delta)}\right)$.*

4.2 Our lower bound

In this section we show that, for every $\beta \geq 0$, there is a temporal graph G of n vertices for which the size of any single-source temporal β -additive spanner of G w.r.t. s is $\Omega\left(\frac{n^2}{1+\beta}\right)$. This gives a lower bound of $\Omega(n^2)$ for the size of a single-source temporal preserver.

The temporal graph G has $n = (13 + \beta)h$ vertices, where h is an integer, and is formed by the union of h pairwise edge-disjoint temporal paths π_1, \dots, π_h . Each path π_i goes from s to a vertex z_i and has length $\Omega(n - i(1 + \beta))$. The construction guarantees that the unique temporal path of G from s to z_i of length of at most $d_G(s, z_i) + \beta$ is π_i . This implies that the size of G is $\Omega\left(\frac{n^2}{1+\beta}\right)$, as desired.

The temporal path π_1 is a Hamiltonian path that spans all the n vertices of G and goes from s to z_1 . All edges of π_1 have time-label 1. The remaining temporal paths are defined recursively. More precisely, for each $i = 2, \dots, h$, the temporal path π_i is defined on top of the temporal path π_{i-1} as follows. Let us number the vertices visited in a traversal of π_{i-1} from s to z_{i-1} in order from 0 to $|\pi_{i-1}| - 1$. The temporal path π_i is defined as a sequence of hops over the vertices of π_{i-1} . We call *offset* a value μ that is equal to $\beta + 7$ for even values of β , and to $\beta + 8$ for odd values of β . The first hop is the one from s to vertex μ ,

if it exists. The rest of the path is given by a maximal alternating sequence of *backward* and *forward* hops that do not visit z_{i-1} . A generic backward hop goes from vertex j , with j odd, to vertex $j - 3$, while a generic forward hop goes from vertex j , with j even, to vertex $j + 5$. All the edges of π_i have time-label i . A pictorial example of the definition of π_i is given in Figure 6. The choice of odd values for the offset is a necessary condition to have pairwise edge-disjoint paths, while the dependency of the offset on β guarantees that π_i is the unique temporal path from s to z_i in G such that $|\pi_i| \leq d_G(s, z_i) + \beta$. Finally, the alternating sequence of backward and forward hops guarantees that $|\pi_i| = \Omega(n - i(1 + \beta))$. The above discussion yields the following theorem, and a corollary for the case $\beta = 0$.

► **Theorem 11.** *For every positive integer n and every $\beta \geq 0$, there is a temporal graph G of n vertices and a source vertex s of G such that any single-source temporal β -additive spanner of G w.r.t. s has size $\Omega\left(\frac{n^2}{1+\beta}\right)$.*

► **Corollary 12.** *For every positive integer n , there is a temporal graph G of n vertices such that any single-source temporal preserver of G w.r.t. s has size $\Theta(n^2)$.*

5 Conclusions

In this paper we addressed the size-stretch trade-offs for temporal spanners. We showed that a temporal clique admits a temporal $(2k - 1)$ -spanner of size $\tilde{O}(n^{1+\frac{1}{k}})$, which implies a spanner having size $\tilde{O}(n)$ and stretch $O(\log n)$. The previous best-known result was the temporal-spanner of [7] which only preserves temporal connectivity between vertices. Our construction guarantees $O(\log n)$ -approximate distances at the cost of only an additional $O(\log n)$ multiplicative factor on the size. We also considered the single-source case for *general* temporal graphs, where we provided almost-tight size-stretch trade-offs, along with the special case of temporal graphs with bounded lifetime.

The main problem that remains open is understanding whether better trade-offs are achievable for temporal cliques. In particular, no superlinear lower bounds are known even for the case of 3-spanners.


Finally, as we already mentioned, temporal graphs admit other natural notions of distances between vertices (which have been used, e.g., in [10, 12, 13]). The most commonly used distances are the *earliest arrival time*, the *latest departure time*, the *fastest time* (i.e., the smallest difference between the arrival and departure time of a temporal path from u to v), and – if each edge has an associated travel time – the *shortest time* distance (i.e., the minimum sum of the travel times of the edges of a temporal path from u to v). One can wonder whether sparse temporal spanners with low stretch are attainable also in the case of the above distances. Unfortunately the answer is negative and strong lower bounds on the size of temporal α -spanners for temporal cliques can be shown even for large values of α , as we discuss in the full version of the paper.

References

- 1 Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen G. Kobourov, and Richard Spence. Graph spanners: A tutorial review. *Comput. Sci. Rev.*, 37:100253, 2020. doi:10.1016/j.cosrev.2020.100253.
- 2 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory Comput. Syst.*, 61(3):907–944, 2017. doi:10.1007/s00224-017-9757-x.
- 3 Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discret. Comput. Geom.*, 9:81–100, 1993. doi:10.1007/BF02189308.

- 4 Kyriakos Axiotis and Dimitris Fotakis. On the size and the approximability of minimum temporally connected subgraphs. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 149:1–149:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.149.
- 5 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.
- 6 Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. 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 167–178. Springer, 2015. doi:10.1007/978-3-662-48350-3_15.
- 7 Arnaud Casteigts, Joseph G. Peters, and Jason Schoeters. Temporal cliques admit sparse spanners. *J. Comput. Syst. Sci.*, 121:1–17, 2021. doi:10.1016/j.jcss.2021.04.004.
- 8 Arnaud Casteigts, Michael Raskin, Malte Renken, and Viktor Zamaraev. Sharp thresholds in random simple temporal graphs. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2022. Full version at arXiv:2011.03738.
- 9 Petter Holme. Temporal networks. In Reda Alhajj and Jon G. Rokne, editors, *Encyclopedia of Social Network Analysis and Mining, 2nd Edition*. Springer, 2018. doi:10.1007/978-1-4939-7131-2_42.
- 10 Silu Huang, Ada Wai-Chee Fu, and Ruifeng Liu. Minimum spanning trees in temporal graphs. In Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 – June 4, 2015*, pages 419–430. ACM, 2015. doi:10.1145/2723372.2723717.
- 11 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, 64(4):820–842, 2002. doi:10.1006/jcss.2002.1829.
- 12 George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019. doi:10.1007/s00453-018-0478-6.
- 13 Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proc. VLDB Endow.*, 7(9):721–732, 2014. doi:10.14778/2732939.2732945.

Resource Sharing Revisited: Local Weak Duality and Optimal Convergence

Daniel Blankenburg  

Research Institute for Discrete Mathematics, Universität Bonn, Germany

Abstract

We revisit the (block-angular) min-max resource sharing problem, which is a well-known generalization of fractional packing and the maximum concurrent flow problem. It consists of finding an ℓ_∞ -minimal element in a Minkowski sum $\mathcal{X} = \sum_{C \in \mathcal{C}} X_C$ of non-empty closed convex sets $X_C \subseteq \mathbb{R}_{\geq 0}^{\mathcal{R}}$, where \mathcal{C} and \mathcal{R} are finite sets. We assume that an oracle for approximate linear minimization over X_C is given.

We improve on the currently fastest known FPTAS in various ways. A major novelty of our analysis is the concept of local weak duality, which illustrates that the algorithm optimizes (close to) independent parts of the instance separately. Interestingly, this implies that the computed solution is not only approximately ℓ_∞ -minimal, but among such solutions, also its second-highest entry is approximately minimal.

Based on a result by Klein and Young [21], we provide a lower bound of $\Omega\left(\frac{|\mathcal{C}|+|\mathcal{R}|}{\delta^2} \log |\mathcal{R}|\right)$ required oracle calls for a natural class of algorithms. Our FPTAS is optimal within this class – its running time matches the lower bound precisely, and thus improves on the previously best-known running time for the primal as well as the dual problem.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Resource sharing, Dantzig-Wolfe-type algorithms, Decreasing minimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.20

Related Version Faster Primal-Dual Convergence for Min-Max Resource Sharing and Stronger Bounds via Local Weak Duality

Full Version: <https://arxiv.org/abs/2111.06820>

Acknowledgements We thank Jens Vygen for many fruitful discussions and Sebastian Pokutta for helpful suggestions.

1 Introduction

1.1 Problem description

Dividing a limited set of *resources* among *customers* is an overarching theme of numerous problems in discrete and continuous optimization. A common formulation of such problems is known as *min-max resource sharing* in the literature. In this work, we consider the *block-angular min-max resource sharing problem* as it was first studied by Grigoriadis and Khachiyan [13]. The problem consists of choosing a feasible resource allocation for every customer, such that the maximum accumulated resource usage is minimized. Formally, it can be described as follows:

There is a finite set of customers \mathcal{C} and a finite set of resources \mathcal{R} . We denote their sizes by $n := |\mathcal{C}|$ and $m := |\mathcal{R}|$. For each customer $C \in \mathcal{C}$, there is a non-empty closed convex set $X_C \subseteq \mathbb{R}_{\geq 0}^m$ of *feasible resource allocations*, also referred to as *block*. The set of *feasible solutions* of the (block-angular) min-max resource sharing problem is given as the Minkowski-sum $\mathcal{X} := \sum_{C \in \mathcal{C}} X_C$.



© Daniel Blankenburg;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 20; pp. 20:1–20:14
Leibniz International Proceedings in Informatics



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

20:2 Resource Sharing Revisited

Further, we assume that we are given, for some constant $\sigma \geq 1$, a σ -approximate *block-solver*, which is an approximate linear minimization oracle for non-negative price vectors. It is specified by functions $f_C : \mathbb{R}_{\geq 0}^m \rightarrow X_C$ for all $C \in \mathcal{C}$ that satisfy

$$\forall y \in \mathbb{R}_{\geq 0}^m : \quad \langle y, f_C(y) \rangle \leq \sigma \text{opt}_C(y), \quad (1)$$

where $\text{opt}_C(y) := \min_{x \in X_C} \langle y, x \rangle$.

The (block-angular) *min-max resource sharing problem* is to compute resource allocations $x_C \in X_C$ for every customer $C \in \mathcal{C}$, such that $x := \sum_{C \in \mathcal{C}} x_C$ attains

$$\lambda^* := \min_{x \in \mathcal{X}} \|x\|_{\infty}. \quad (2)$$

We abbreviate this problem as *resource sharing problem* in the following. In this work, we consider fully polynomial-time approximation schemes relative to σ . For $\delta > 0$, we construct a solution $x \in \mathcal{X}$ with $\|x\|_{\infty} \leq \sigma(1 + \delta)\lambda^*$ within a number of oracle calls that is polynomial in n, m , and δ^{-1} .

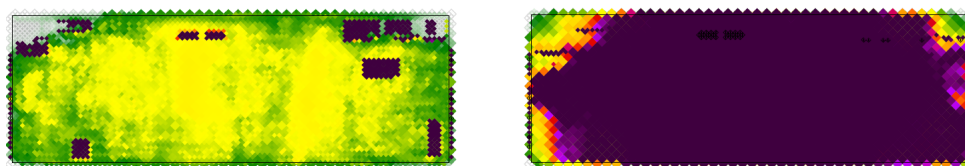
Algorithms that interact with the feasible region only via a linear minimization oracle are known as algorithms of the *Dantzig-Wolfe type*. An iteration consists of choosing a price vector $y \in \mathbb{R}_{\geq 0}^m$ and querying the linear minimization oracle of a customer $C \in \mathcal{C}$ with y . The computed solution is a convex combination of the solutions returned from the oracle. At their core, Dantzig-Wolfe-type algorithms are primal-dual algorithms. In the case of the resource sharing problem, the dual is to find $q \in \Delta_m := \{p \in [0, 1]^m : \|p\|_1 = 1\}$, such that $\min_{x \in \mathcal{X}} \langle q, x \rangle = \max_{p \in \Delta_m} \min_{x \in \mathcal{X}} \langle p, x \rangle$. Strong duality is implied by von Neumann's minimax theorem:

$$\max_{p \in \Delta_m} \min_{x \in \mathcal{X}} \langle p, x \rangle = \min_{x \in \mathcal{X}} \max_{p \in \Delta_m} \langle p, x \rangle = \min_{x \in \mathcal{X}} \|x\|_{\infty} = \lambda^*. \quad (3)$$

The resource sharing formulation can be used to model a large variety of packing problems in combinatorial optimization. Prominent examples include multicommodity and concurrent flow problems, where the linear minimization tasks correspond to shortest path problems, or fractional Steiner tree packing problems, where the linear minimization tasks correspond to minimum weight Steiner tree problems and can be implemented, for example, with a fast 2-approximation.

Using a linear minimization oracle to interact with the feasible region is a standard tool in convex optimization, most often encountered in the form of conditional gradient descent (a.k.a. Frank-Wolfe algorithm [9]), that has regained interest in current research as it leads to surprisingly fast algorithms in practice [17, 19]. Apart from efficiency, utilizing a linear minimization oracle has the advantage that solutions are constructed explicitly as an (often sparse) combination of extreme points. This is highly useful if one is interested in an integral solution to the packing problem, since, in practice, a close to optimal integral solution can often be recovered by applying e.g. randomized rounding to the sparse fractional solution [28].

Resource sharing has a long history of incremental generalizations and improvements since Shahrokhi and Matula [29] introduced the idea to use a linear optimization oracle with an exponential weight function in the context of multicommodity flows. At their heart, most, if not all, steps in this line of work can be interpreted as variants of the now well-known multiplicative weight update method. Since we also present a variant of that algorithm and improve on the best-known running time in this setting, our work can be regarded as another step in that line. But more than that, we study an aspect that – to the best of our knowledge – is absent from any treatment of multiplicative-weight-based algorithms even in special cases such as multicommodity flow. We are interested in the stability of such algorithms and in



(a) High congestion only in local hotspots. (b) High congestion in almost every part.

■ **Figure 1** Comparison of two fractional routing results. Congestion of global routing tiles is visualized by color (lowest congestion in green, highest in violet).

deriving guarantees that go beyond the min-max objective. For example, one might alter a given instance \mathcal{X} by adding a completely independent resource and consider $\mathcal{X} \times \{\Lambda\}$ with $\Lambda \gg \min_{x \in \mathcal{X}} \|x\|_\infty$. In this case, ℓ_∞ -guarantees of existing algorithms do not provide any insight into the quality of their computed solution on the perturbed instance when restricted to \mathcal{X} . We introduce techniques that allow a comprehensive treatment of such situations.

To provide more motivation for this pursuit, we consider the example of global routing in VLSI design, where a resource sharing formulation has proven successful in theory and practice [12, 16, 24]. Highly simplified, the problem consists of a Steiner tree packing problem in a grid graph. One seeks to find a collection of (fractional) Steiner trees that minimize the maximum edge overload. Figure 1 depicts two fractional routing results on an industrial microprocessor. The maximum edge congestion is the same in both cases, so both solutions have the same objective value w.r.t. the resource sharing problem. As indicated in Figure 1a, however, the maximum congestion might be caused by local effects that are beyond reach for global optimization. A practicable algorithm must be resilient to such local hotspots and produce solutions that are close to optimal on the set of remaining resources. The solution in Figure 1a is clearly to be preferred over that in Figure 1b. In practice, the ideal outcome might be a solution $x \in \mathcal{X}$ that minimizes the maximal entry, and among such solutions, it minimizes the second-highest entry, and among those, it minimizes the third-highest, and so on. This concept is known as *decreasingly minimal* [6].

► **Definition 1.** Let $x \in \mathbb{R}^m$. We denote by x_\downarrow the vector x with entries sorted in decreasing order. We introduce the decreasing preorder, by defining for $x, y \in \mathbb{R}^m$, $x \leq_{dec} y$ if either $x_\downarrow = y_\downarrow$ or there exists $k \in \{1, \dots, m\}$ such that $(x_\downarrow)_k < (y_\downarrow)_k$ and $(x_\downarrow)_j = (y_\downarrow)_j$ for all $j < k$. Given a set $X \subseteq \mathbb{R}^m$, we say that an element $x \in X$ is decreasingly minimal if $x \leq_{dec} y$ holds for all $y \in X$.

1.2 Our contribution

In this work, we present a fully polynomial-time approximation scheme for the primal as well as the dual of the resource sharing problem. Our algorithm is an extension of the algorithm by Müller, Radke, and Vygen [24].

We introduce the novel concept of *local weak duality* (Definition 2) to analyze the *core algorithm* (Algorithm 1) which is a variant of the multiplicative weight update method. Given local weak duality, we can prove stronger bounds on $\max_{r \in S} x_r$ for the computed solution $x \in \mathcal{X}$ and certain subsets $S \subseteq \mathcal{R}$. That provides a theoretical counterpart to the empirical observation that the algorithm optimizes (close to) independent parts of the instance separately. Moreover, it allows concluding that – with an exact block solver – our algorithm always computes a solution that is approximately decreasingly minimal on the two highest entries. Such guarantees were not known even for special cases of the problem and

could, in principle, be transferred to many other settings where the multiplicative weight update method is employed. We also provide a negative result, namely that our algorithm will not compute close to decreasingly minimal solutions on the three highest entries in general.

Furthermore, we present an iterative scaling scheme in conjunction with an amortized running time analysis that results in the currently fastest known constant factor approximation for the resource sharing problem with $\mathcal{O}((n+m) \log m)$ many oracle calls. This is then utilized as an initial scaling technique to design an FPTAS improving on the best-known running time for the resource sharing problem. Also, dual convergence, which follows immediately with our analysis, was not shown in prior work.

Moreover, we discuss the limits of algorithms in this setting. We study a class of natural extensions of the core algorithm and prove, using a result by Klein and Young [21], that any algorithm from this class requires $\Omega(\frac{n+m}{\delta^2} \log m)$ oracle calls to compute a $(1+\delta)$ -approximate solution (for a range of parameters as described in Theorem 13). This matches the running time of our algorithm precisely. As this is independent of the choice of the prices, this proves that – in a certain sense – multiplicative price updates are optimal, and that no warm-start analysis (i.e. reducing the running time by starting with a close to optimal dual solution) of such algorithms is possible.

1.3 Related work

The resource sharing problem originates from the maximum concurrent flow problem. For this special case, Shahrokhi and Matula [29] introduced the idea to use Dantzig-Wolfe-type algorithms with an exponential weight function. Another important special case of this problem is *fractional packing*, which can be described as follows. Given a polyhedron $P \subseteq \mathbb{R}^k$, a matrix $A \in \mathbb{R}^{m \times k}$ that satisfies $Ax \geq 0$ for all $x \in P$, and a vector $b \in \mathbb{R}_{>0}^m$, find an $x \in P$ that satisfies $Ax \leq b$. The width of the instance is defined as $\rho := \max_{x \in P} \max_{i=1, \dots, m} (Ax)_i / b_i$. Plotkin, Shmoys, and Tardos [26] studied this problem in the context of Dantzig-Wolfe-type algorithms. Their results were generalized and improved multiple times [5, 11, 13, 18, 20, 22, 24]. An important idea in this line of work is a step-size technique introduced by Garg and Könemann [11] and extended by Fleischer [5], which is used to design algorithms with width-independent running times.

The general version of the block-angular min-max resource sharing problem, as it is the subject of this work, was studied first by Grigoriadis and Khachiyan [13]. In their formulation, one is given non-empty closed convex blocks B_C for each $C \in \mathcal{C}$, and resource allocation functions $g^{(C)} : B_C \rightarrow \mathbb{R}_{\geq 0}^m$, which are convex and non-negative in each coordinate. Then, one seeks to compute $\min\{\max_{r \in \mathcal{R}} \sum_{C \in \mathcal{C}} g^{(C)}(b_C)_r : b_C \in B_C \forall C \in \mathcal{C}\}$. Their optimization oracle solves $\min_{b_C \in B_C} \langle y, g^{(C)}(b_C) \rangle$ for a given price vector $y \in \mathbb{R}_{\geq 0}^m$. It is easy to see that this formulation fits into our framework by defining X_C as the convex hull of $g^{(C)}(B_C)$. The currently fastest known algorithm for the general case is due to Müller, Radke, and Vygen [24]. They present a sequential algorithm that computes a solution of value at most $\sigma(1+\delta)$ within $\mathcal{O}((n+m) \log m (\delta^{-2} + \log \log m))$ oracle calls. All of the mentioned approaches can be interpreted as variants of the multiplicative weights update method [2].

Klein and Young [21] studied lower bounds on the number of iterations that are required by any Dantzig-Wolfe-type algorithm to compute a $(1+\delta)$ -approximate solution for fractional packing. They provide an asymptotic width-dependent bound of $\Omega\left(\min\left\{\rho \frac{\log m}{\delta^2}, m^{1/2-\gamma}\right\}\right)$ (for any fixed $\gamma \in (0, 1/2)$). This matches known upper bounds precisely for a range of parameters. If the width of the instance is unbounded, it is easy to see that $\Omega(m)$ oracle

■ **Algorithm 1** Core Resource Sharing Algorithm with parameters ϵ, T .

```

1:  $y \leftarrow \mathbb{1} \in \mathbb{R}^m$ 
2: for  $t = 1, \dots, T$  do
3:   for  $C \in \mathcal{C}$  do
4:      $\alpha \leftarrow 0$ 
5:      $s_C^{(t)} \leftarrow 0$ 
6:     while  $\alpha < 1$  do
7:        $b \leftarrow f_C(y)$ 
8:        $\xi \leftarrow \min\{1 - \alpha, 1/\|b\|_\infty\}$ 
9:        $y_r \leftarrow y_r \exp(\epsilon \xi b_r) \quad \forall r \in \mathcal{R}$ 
10:       $s_C^{(t)} \leftarrow s_C^{(t)} + \xi b$ 
11:       $\alpha \leftarrow \alpha + \xi$ 
12:    end while
13:  end for
14:   $s^{(t)} \leftarrow \sum_{C \in \mathcal{C}} s_C^{(t)}$ 
15: end for
16: return  $\frac{1}{T} \sum_{t=1}^T s^{(t)}$ 

```

calls are required to compute a constant-factor approximation [14]. If one is not restricted to algorithms of the Dantzig-Wolfe-type, then it is known how to avoid the δ^{-2} dependence on the running time for the case of fractional packing [1, 3].

Decreasingly minimal solutions to optimization problems appear under many different names in the literature, such as lexicographically optimal [10, 23], egalitarian [4], fair [8, 27] and more recently in a line of work by Frank and Murota – who study the integral case – as decreasingly minimal [6, 7]. We are going to use their notation in the following. It is known how to find such solutions with linear programming techniques [25, 27]. In our case, since \mathcal{X} is a non-empty closed convex subset of $\mathbb{R}_{\geq 0}^m$, it contains a unique decreasingly minimal element [27]. A concept related to decreasing minimization is that of *majorization* [15]. If the set of feasible solutions contains a least majorized element, it is also the decreasingly minimal element and can be extracted as the minimum of non-decreasing separable convex functions [30]. We are not aware of results w.r.t. decreasingly minimal solutions in the context of Dantzig-Wolfe-type algorithms.

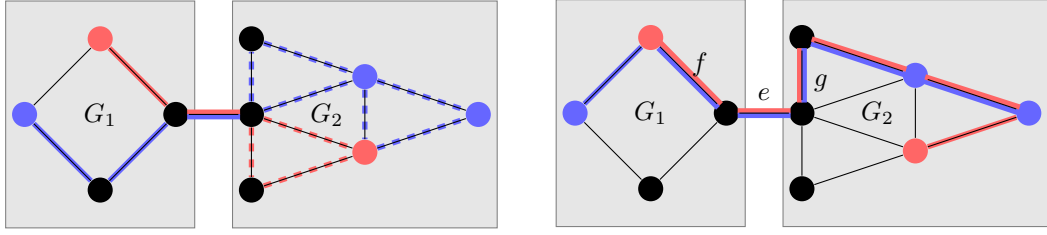
1.4 Outline

The core algorithm (Algorithm 1, [24]) starts with the uniform price vector $y = \mathbb{1} \in \mathbb{R}^m$ and, after every oracle call, it updates the prices $y_r \leftarrow y_r \exp(\epsilon \xi b_r)$, where $\epsilon > 0$ is a fixed parameter of the algorithm. It runs in $T \in \mathbb{N}$ phases (iterations of the outer loop). At the end, the average solution over all phases $x^{(T)} := \frac{1}{T} \sum_{t=1}^T s^{(t)}$ is returned. The restricted step sizes in Line 8 were first proposed by Garg and Könemann [11] in the case of multicommodity flows.

The standard tool to prove primal convergence of algorithms that are based on the multiplicative weight update method is weak duality: For $y \in \mathbb{R}_{\geq 0}^m$, it holds that

$$\sum_{C \in \mathcal{C}} \langle y, f_C(y) \rangle \leq \sigma \min_{x \in \mathcal{X}} \langle y, x \rangle \leq \sigma \|y\|_1 \min_{x \in \mathcal{X}} \|x\|_\infty = \sigma \|y\|_1 \lambda^*. \quad (4)$$

This is sufficient to derive bounds on $\|x^{(T)}\|_\infty$. However, we aim to prove stronger bounds on $\max_{r \in S} x_r^{(T)}$ for certain subsets of resources $S \subseteq \mathcal{R}$. To this end, we introduce the concept of *local weak duality*, which generalizes weak duality. We briefly describe the intuition behind



(a) The decreasingly minimal solution λ with a decomposition $\lambda = \lambda^{(blue)} + \lambda^{(red)}$. Here $\lambda^{(blue)}$ ($\lambda^{(red)}$) assigns value 1 to all edges that are marked with solid blue (red) lines and value $1/2$ to all edges that are marked with dashed blue (red) lines.

(b) Example of a bad oracle. Consider edge prices $y \in \mathbb{R}_{\geq 0}^{E(G)}$, given by $y_e = 2$, $y_f = y_g = 1$, and zero otherwise. Then, a 2-approximation oracle may return the solution above, not satisfying local weak duality on $E(G_1)$ or $E(G_2)$ for any value $\mu < 2$.

■ **Figure 2** An instance of fractional Steiner tree packing in a graph G that consists of two subgraphs G_1 and G_2 which are joined by an edge. The goal is to find fractional Steiner trees that connect the blue, respectively red terminals, such that the maximum edge load is minimized.

this notion. Given $y \in \mathbb{R}_{\geq 0}^m$, one may consider the *local* objective value $\sum_{C \in \mathcal{C}} \sum_{r \in S} y_r f_C(y)_r$ of the oracles on S (e.g. the cost of the paths restricted to a subset of edges in the multicommodity flow case). A local analog to weak duality is given if this objective value can be bounded by $\mu \sum_{r \in S} y_r$ for some $\mu > 0$, independently of y_r for $r \in \mathcal{R} \setminus S$ (the prices on the remaining edges). The following definition includes a different price vector for every customer, which is necessary to deal with the sequential price updates of the core algorithm. Then, the upper bound on the local objective value is defined using the point-wise maximum of these prices.

► **Definition 2.** We say that an instance \mathcal{X} of the resource sharing problem satisfies local weak duality w.r.t. a subset of resources $S \subseteq \mathcal{R}$ and $\mu \geq 0$ if for any collection of non-negative price vectors $(y^{(C)})_{C \in \mathcal{C}} \subseteq \mathbb{R}_{\geq 0}^m$ it holds that

$$\sum_{C \in \mathcal{C}} \sum_{r \in S} y_r^{(C)} f_C(y^{(C)})_r \leq \mu \sum_{r \in S} \max_{C \in \mathcal{C}} y_r^{(C)}. \quad (5)$$

This definition generalizes weak duality, in the sense that every instance satisfies local weak duality w.r.t. $S = \mathcal{R}$ and $\mu = \sigma \lambda^*$.

Let us briefly present an exemplary application. Figure 2 displays an instance of the fractional Steiner tree packing problem with unit capacities. The goal is to find fractional Steiner trees in G that connect the blue, respectively the red terminals such that the maximum edge usage is minimized. We have $\mathcal{C} = \{blue, red\}$ and $\mathcal{R} = E(G)$. The figure also depicts an optimum solution $\lambda = \lambda^{(red)} + \lambda^{(blue)}$ to the resource sharing problem as described in the caption of Figure 2a. In this case, λ is also the decreasingly minimal solution. As the figure indicates, G can be decomposed into two subgraphs G_1 and G_2 that are joined by an edge e . It is clear that e is the bottleneck, meaning that $\lambda_e = \min_{x \in \mathcal{X}} \|x\|_\infty = 2$. However, the (local) maximum usage on G_1 and G_2 is lower, it holds $\max_{e \in E(G_1)} \lambda_e = 1$ and $\max_{e \in E(G_2)} \lambda_e = \frac{1}{2}$. We observe that the Steiner tree problems decompose into two independent subproblems in G_1 and G_2 , i.e. the blue, respectively red terminals in G_1 and G_2 need to be connected to the contained endpoint of e . Let us discuss two examples of how to apply local weak duality to this instance for different oracles.

- (i) The oracle is exact. Thus it also solves the subproblems in G_1 and G_2 exactly. Now consider price vectors $y^{(red)}, y^{(blue)} \in \mathbb{R}_{\geq 0}^{E(G)}$. Since $\lambda^{(blue)}$ and $\lambda^{(red)}$ are feasible solutions for the *blue*, respectively the *red* customer, we get:

$$\sum_{e \in E(G_1)} y_e^{(blue)} f_{blue}(y^{(blue)})_e \leq \sum_{e \in E(G_1)} y_e^{(blue)} \lambda_e^{(blue)} \leq \sum_{e \in E(G_1)} \lambda_e^{(blue)} \max\{y_e^{(blue)}, y_e^{(red)}\}.$$

Applying the analogous inequality to the red customer and using that $\lambda = \lambda^{(blue)} + \lambda^{(red)}$, we can derive

$$\sum_{e \in E(G_1)} y_e^{(blue)} f_{blue}(y^{(blue)})_e + \sum_{e \in E(G_1)} y_e^{(red)} f_{red}(y^{(red)})_e \leq \sum_{e \in E(G_1)} \lambda_e \max\{y_e^{(blue)}, y_e^{(red)}\}.$$

The right hand side can be bounded by $\max_{e \in E(G_1)} \lambda_e \sum_{e \in E(G_1)} \max\{y_e^{(blue)}, y_e^{(red)}\}$. Therefore, local weak duality is satisfied w.r.t. $S = E(G_1)$ and $\mu = \max_{e \in E(G_1)} \lambda_e = 1$. The very same argument can be applied to show that local weak duality is also satisfied w.r.t. $S = E(G_2)$ and $\mu = \max_{e \in E(G_2)} \lambda_e = \frac{1}{2}$.

- (ii) The oracle is a 2-approximation given by a path-decomposition algorithm. In this case, we know that it will solve the subproblems in G_1 exactly (as these are shortest path problems) and the approximation guarantee of 2 also holds for the subinstance in G_2 . Thus, with the same argument as above, local weak duality is satisfied with regard to $S = E(G_1)$ and $\mu = 1$, and with regard to $S = E(G_2)$ and $\mu = 2 \cdot \frac{1}{2} = 1$.

In general, we cannot assume local weak duality with $\mu < 2$ for all 2-approximation oracles as described in Figure 2b for this instance. This illustrates that our notion of local weak duality is not only a property of the convex region that we consider but also a property of the oracle. This instance is an example of a product case, which we briefly discuss in Section 3.

In Section 2.1, we analyze the core algorithm with new techniques. We prove primal-dual convergence and that, under local weak duality, $\max_{r \in S} x_r^{(T)}$ is bounded by μ plus an additive error that is linear in δ . Dual convergence was not shown in [24]. The core algorithm is sufficient to obtain fast convergence in normalized settings, i.e. when λ^* is known up to a constant factor. To derive a FPTAS for the general problem one needs to extend this algorithm by a prior constant factor approximation. In [24] this is done with a scaling/binary search approach. We present a faster constant-factor approximation in Section 2.2. This allows to transfer all results from the normalized to the general setting without increasing the asymptotic number of oracle calls. We summarize this in the following main theorem.

► **Theorem 3 (Main Theorem).** *Let \mathcal{X} be an instance of the resource sharing problem and $\delta \in (0, 1]$. One can compute a primal solution $x \in \mathcal{X}$ and a dual solution $z \in \Delta_m$ satisfying*

$$\|x\|_\infty \leq (1 + \delta)\sigma\lambda^*, \quad \min_{x \in \mathcal{X}} \langle z, x \rangle \geq (1 - \delta) \frac{\lambda^*}{\sigma}, \quad (6)$$

and, for all $S \subseteq \mathcal{R}$, $\mu \geq 0$ such that (5) holds,

$$\max_{r \in S} x_r \leq \mu + \delta \max\{\lambda^*, \mu\}, \quad (7)$$

using $\mathcal{O}\left(\frac{n+m}{\delta^2} \log m\right)$ oracle calls, and further operations taking polynomial time.

In Section 3, we show how to apply this result to product settings and that the computed solution is close to decreasingly minimal on the two highest entries in the following sense.

► **Corollary 4.** *Let $\lambda \in \mathcal{X}$ be decreasingly minimal. If the block solver is exact, i.e. $\sigma = 1$, then the returned solution $x \in \mathcal{X}$ satisfies*

$$(x_{\downarrow})_1 \leq (\lambda_{\downarrow})_1 + \delta\lambda^* \quad \text{and} \quad (x_{\downarrow})_2 \leq (\lambda_{\downarrow})_2 + \delta\lambda^*. \quad (8)$$

This analysis is best possible for the general case. In the full version we show that an analogous version of Corollary 4 does not hold for the three highest entries. Furthermore, it is not possible to reduce the additive constant of $\delta\lambda^*$ to $\delta(\lambda_{\downarrow})_2$ in the second inequality of (8). In a sense, an additive error of $\mathcal{O}(\delta\lambda^*)$ is best possible in our framework, because the core algorithm is known to make an additive error of $\mathcal{O}(\delta)$ and we scale the instance such that $\lambda^* \in [c, C]$ for some constants $0 < c < C$ before applying it.

In Section 4, we study a class of generalizations of the core algorithm. These can be described as standard block-coordinate descent algorithms. This class contains all algorithms that run in a number of phases T , and in each phase $1, \dots, T$ construct a solution for every customer by using *any* choice of prices and the restricted step size rule as in Line 8 of the core algorithm. Our algorithm is optimal within this class in the sense that any such algorithm requires $\Omega(\frac{n+m}{\delta^2} \log m)$ many oracle calls to construct a $(1 + \delta)$ -approximate solution (Theorem 13), even if $\lambda^* = 1$ is known. This follows from a result by Klein and Young [21] and matches the upper bound provided by our algorithm precisely.

2 Proof of the main theorem

2.1 Analysis on normalized instances

First, we prove the main theorem for normalized instances, meaning those for which $\lambda^* \in [c, 1]$ is known for a constant $c > 0$. Later, in Section 2.2, we show how to remove this assumption. We write $x^{(t)} := \frac{1}{t} \sum_{p=1}^t s^{(p)} \in \mathcal{X}$ for the current solution and $y^{(t)}$ for the price vector y after phase t of the core algorithm is completed. Note that $y_r^{(t)} = \exp(\epsilon \sum_{p=1}^t s_r^{(p)}) = \exp(\epsilon t x_r^{(t)})$ holds for all $r \in \mathcal{R}$ and $t = 1, \dots, T$. This can be used to deduce a simple bound for any $S \subseteq \mathcal{R}$:

$$\max_{r \in S} x_r^{(t)} = \max_{r \in S} \frac{1}{\epsilon t} \log y_r^{(t)} \leq \frac{1}{\epsilon t} \log \sum_{r \in S} y_r^{(t)}. \quad (9)$$

Especially, $\|x^{(t)}\|_{\infty} \leq \frac{1}{\epsilon t} \log \|y^{(t)}\|_1$ holds. We denote the dual objective values that correspond to the price vectors $y^{(t)}$ by

$$\Theta_t := \min_{x \in \mathcal{X}} \frac{\langle y^{(t)}, x \rangle}{\|y^{(t)}\|_1} = \frac{\sum_{C \in \mathcal{C}} \text{opt}_C(y^{(t)})}{\|y^{(t)}\|_1}. \quad (10)$$

We provide a bound that relates the number of oracle calls to the price vector $y^{(t)}$. This bound improves (slightly) on the bound that was shown in [24], which was $tn + \frac{m}{\epsilon} \log \|y^{(t)}\|_1$.

► **Lemma 5.** *The number of oracle calls of the core algorithm until termination of phase $t = 1, \dots, T$ is bounded by*

$$tn + \frac{m}{\epsilon} \log \frac{\|y^{(t)}\|_1}{m}. \quad (11)$$

A proof can be found in the full version. The following lemma states a bound on $\sum_{r \in S} y_r^{(t)}$, which implies a bound on $\max_{r \in S} x_r^{(t)}$ according to Inequality (9). In the case $S = \mathcal{R}$, this provides a bound on the primal error and the number of oracle calls. Again, we refer to the full version for a proof of the lemma.

► **Lemma 6** (Upper bound on the price increase). *Let \mathcal{X} be an instance of the resource sharing problem that satisfies local weak duality w.r.t. $S \subseteq \mathcal{R}$ and $\mu > 0$. Let $\eta := \exp(\epsilon) - 1$. Assume that $\eta\mu < 1$. Then for all $t = 0, 1, \dots, T$, it holds that*

$$\sum_{r \in S} y_r^{(t)} \leq |S| \exp\left(\frac{t\eta\mu}{1 - \eta\mu}\right). \quad (12)$$

As pointed out earlier, every instance satisfies local weak duality w.r.t. \mathcal{R} and $\sigma\lambda^*$. So the bound $\|y^{(t)}\|_1 \leq m \exp(t\eta\sigma\lambda^*/(1 - \eta\sigma\lambda^*))$ that was stated in [24] follows. In this case, however, it is possible to insert the definition of the dual values to obtain the primal-dual bound $\|y^{(t)}\|_1 \leq m \exp\left(\frac{\eta\sigma}{1 - \eta\sigma\lambda^*} \sum_{p=1}^t \Theta_p\right)$, which can be used to prove also dual convergence of the average dual solution.

► **Theorem 7** (Bound on the primal and dual error). *Assume $\epsilon \in (0, 1]$, $\eta\sigma\lambda^* < 1$. For every $t = 1, \dots, T$ the primal solution $x^{(t)}$ and the average dual solution $z^{(t)} := \frac{1}{t} \sum_{p=1}^t \frac{y^{(p)}}{\|y^{(p)}\|_1} \in \Delta_m$ satisfy*

$$\|x^{(t)}\|_\infty \leq \frac{\log m}{\epsilon t} + \frac{1 + \epsilon}{1 - \eta\sigma\lambda^*} \sigma\lambda^*, \quad \min_{x \in \mathcal{X}} \langle z^{(t)}, x \rangle \geq \frac{1 - \eta\sigma\lambda^*}{\sigma(1 + \epsilon)} \left(\lambda^* - \frac{\log m}{\epsilon t} \right). \quad (13)$$

Moreover, if \mathcal{X} satisfies local weak duality w.r.t. $S \subseteq \mathcal{R}$ and $\mu > 0$, then

$$\max_{r \in S} x_r^{(t)} \leq \frac{\log |S|}{\epsilon t} + \frac{1 + \epsilon}{1 - \eta\sigma\lambda^*} \mu. \quad (14)$$

For normalized instances, the main theorem follows with a straightforward calculation from Theorem 7 by choosing $\epsilon = \frac{\delta}{8\sigma}$ and $T = \left\lceil \frac{\log m}{2\sigma\epsilon^2} \right\rceil$. Omitted proofs can be found in the full version.

2.2 Constant-factor approximation in $\mathcal{O}((n + m) \log m)$ oracle calls

In the following, we assume $\lambda^* \in [1, \sigma m]$, which can be guaranteed in n oracle calls, by computing for each customer a solution to the uniform price vector $x := \sum_{C \in \mathcal{C}} f_C(\mathbb{1}) \in \mathcal{X}$ and scaling the instance with $\frac{\sigma m}{\|x\|_\infty}$. Our constant-factor approximation, Algorithm 2, is similar to the core algorithm, but it works with adaptive parameters, may discard the solutions of some phases, and restart them. This is done by checking a bound on the sum of the prices, in Line 13, after every oracle call. The algorithm maintains a guess on λ^* , denoted by Λ , which influences the convex coefficient ξ . A violation of the bound indicates that the guess was too low. This leads to a restart of the phase with $\epsilon \leftarrow \epsilon/2$ and $\Lambda \leftarrow 2\Lambda$.

Let $K^* := \lceil \log \lambda^* \rceil$. We denote by K the number of restarts of the algorithm (i.e. times the if-statement in Line 13 is satisfied). Further, $t_1 \leq \dots \leq t_K$ denote the (not necessarily distinct) indices of the phases in which the restarts occur. For a phase t , we write $\epsilon^{(t)}$ for the ϵ -value with which it was completed successfully. As before, we write $x^{(t)} := \frac{1}{t} \sum_{p=1}^t s^{(p)}$ for the current solution after phase t . By construction of the algorithm, we have that $y_r^{(t)} = \exp\left(\sum_{p=1}^t \epsilon^{(p)} s_r^{(p)}\right)$. Note that $\epsilon^{(p)}$ is decreasing for $p = 1, \dots, t$. Therefore, one can bound the maximum usage of the current solution at termination of phase t analogous to previously by $\|x^{(t)}\|_\infty \leq \frac{1}{\epsilon^{(t)} t} \log \|y^{(t)}\|_1$. We only sketch the analysis of the algorithm in the following. Note that after the k 'th restart, $\epsilon = \frac{1}{2^{k4\sigma}}$. So, for larger k , the price updates $y_r \leftarrow \exp(\epsilon \xi b_r)$ get smaller and smaller. Similarly to the proof of Lemma 6, one can show that after K^* restarts, ϵ is small enough to guarantee that the price bound will not be violated again.

■ **Algorithm 2** Constant-factor approximation for the resource sharing problem.

```

1:  $y \leftarrow \mathbb{1} \in \mathbb{R}^m$ 
2:  $\epsilon \leftarrow \frac{1}{4\sigma}$                                 ▷ Current epsilon for the price update
3:  $\Lambda \leftarrow 1$                                 ▷ Current guess on  $\lambda^*$ 
4: for  $t = 1, \dots, T := \lceil \log m \rceil$  do
5:    $y^{(t-1)} \leftarrow y$                             ▷ Store the last price vector
6:   for  $C \in \mathcal{C}$  do
7:      $\alpha \leftarrow 0, s_C^{(t)} \leftarrow 0$ 
8:     while  $\alpha < 1$  do
9:        $b \leftarrow f_C(y)$ 
10:       $\xi \leftarrow \min\{1 - \alpha, \Lambda / \|b\|_\infty\}$     ▷ Decide the ‘‘amount’’ with which  $b$  is taken
11:       $y_r \leftarrow y_r \exp(\epsilon \xi b_r) \quad \forall r \in \mathcal{R}$ 
12:       $s_C^{(t)} \leftarrow s_C^{(t)} + \xi b, \alpha \leftarrow \alpha + \xi$ 
13:      if  $\|y\|_1 > m \exp(t)$  then                    ▷ Check price bound
14:         $y \leftarrow y^{(t-1)}$                         ▷ Reset the prices to those of the start of this phase
15:         $\epsilon \leftarrow \epsilon/2, \Lambda \leftarrow 2\Lambda$     ▷ Reduce  $\epsilon$ , increase the guess on  $\lambda^*$ 
16:        go to 6                                        ▷ Restart phase, solutions from this phase are discarded
17:      end if
18:    end while
19:  end for
20:   $s^{(t)} \leftarrow \sum_{C \in \mathcal{C}} s_C^{(t)}$ 
21: end for
22: return  $\frac{1}{T} \sum_{t=1}^T s^{(t)}$ 

```

► **Lemma 8.** *The total number of restarts K is bounded by K^* .*

This means that the number of restarts is bounded by $\lceil \log \lambda^* \rceil \leq \lceil \log \sigma m \rceil \in \mathcal{O}(\log m)$. Analogously to Lemma 5, one can deduce the following bound on the number of oracle calls, which depends on the indices of the phases that are restarted.

► **Lemma 9.** *There is a constant $c_1 > 0$, such that the number of oracle calls is bounded by $Tn + c_1 m \left(T + \sum_{i=1}^K t_i \right)$.*

To use this bound one exploits that (due to the price bound in Line 13)

$$\lambda^* \leq \|x^{(t)}\|_\infty \leq \frac{1}{\epsilon^{(t)} t} \log \|y^{(t)}\|_1 \leq \frac{\log m}{\epsilon^{(t)} t} + \frac{1}{\epsilon^{(t)}} \quad (15)$$

holds for every successfully completed phase t . Thus if $\lambda^* \gg 1$, then $\epsilon^{(t)}$ must be small already for low values of t . This implies that many restarts occurred in the early phases. Indeed, we provide upper bounds on the t_i , which allow estimating $\sum_{i=1}^K t_i$ by a geometric series resulting in $\sum_{i=1}^K t_i \in \mathcal{O}(\log m)$.

► **Lemma 10.** *There are constants $c_2, c_3 > 0$, such that $t_i \leq c_2 + 2^{i-K^*} c_3 \log m$ holds $\forall i = 1, \dots, K$. Thus the number of oracle calls is in $\mathcal{O}((n+m) \log m)$.*

Lemma 8 states that phase T is completed with $\epsilon^{(T)} \geq \frac{1}{2^{K^*} 4\sigma} \geq \frac{1}{\lambda^* 8\sigma}$. Inserting this into Inequality (15) shows that $\|x^{(T)}\|_\infty \leq 16\sigma\lambda^*$. The analysis is summarized in the following theorem.

► **Theorem 11.** *Algorithm 2 computes a solution $x^{(T)} \in \mathcal{X}$ to the resource sharing problem that satisfies $\|x^{(T)}\|_\infty \leq 16\sigma\lambda^*$ using $\mathcal{O}((n+m) \log m)$ oracle calls.*

Combining Algorithm 2 with the core algorithm proves the main theorem. Omitted proofs from this section can be found in the full version.

3 Decreasing Minimality on the two highest entries

In this section, we discuss the implications of local weak duality. In particular, we prove that our algorithm computes solutions that are close to decreasingly minimal on the two largest entries. Before that, let us consider a simple application, which is the *product case*. Assume that the instance of the resource sharing problem consists of multiple independent parts, that is, there exists a (perhaps unknown) partition of the resources $\mathcal{R} = \mathcal{R}_1 \cup \dots \cup \mathcal{R}_k$ on which the customers act “independently”. More precisely, we assume that each of the convex sets X_C can be decomposed into a product $X_C = X_1^{(C)} \times \dots \times X_k^{(C)}$ where $X_i^{(C)} \subseteq \mathbb{R}_{\geq 0}^{|\mathcal{R}_i|}$ for $i = 1, \dots, k$. Let us write $\mathcal{X}_i := \sum_{C \in \mathcal{C}} X_i^{(C)}$ for $i = 1, \dots, k$. Note that we have $\mathcal{X} = \mathcal{X}_1 \times \dots \times \mathcal{X}_k$. Let $C \in \mathcal{C}$ and $y^{(C)} \in \mathbb{R}_{\geq 0}^m$. In this case, \mathcal{X} satisfies local weak duality w.r.t. \mathcal{R}_j and $\min_{x_j \in \mathcal{X}_j} \|x_j\|_\infty$ for every $j = 1, \dots, k$. A proof can be found in the full version. So, indeed, according to the main theorem, Theorem 3, every independent part of the instance is optimized separately, i.e. the primal solution $x^{(T)}$ satisfies $\max_{r \in \mathcal{R}_i} x_r^{(T)} \leq \min_{x_i \in \mathcal{X}_i} \|x_i\|_\infty + \delta \lambda^*$ for all $i = 1, \dots, k$. We already saw a natural example of a product setting in Figure 2. As discussed in the beginning, especially in these product cases it is reasonable to assume local weak duality also for approximate block solvers (with their approximation guarantee). A more surprising observation is that any instance \mathcal{X} with an exact block solver satisfies local weak duality w.r.t. to all but one coordinate and the value of the second-highest entry in the decreasingly minimal solution.

► **Lemma 12.** *Let \mathcal{X} be an instance of the resource sharing problem with $\sigma = 1$. Let $\lambda \in \mathcal{X}$ be the decreasingly minimal element. Assume that $(\lambda_\downarrow)_1 > (\lambda_\downarrow)_2$ holds. Let $r^* \in \mathcal{R}$ be the coordinate of the unique maximum entry, i.e. $\lambda_{r^*} = (\lambda_\downarrow)_1$. Then \mathcal{X} satisfies local weak duality w.r.t. $S := \mathcal{R} \setminus \{r^*\}$ and $\mu := (\lambda_\downarrow)_2$.*

A proof of this statement can be found in the full version. Note that if $(\lambda_\downarrow)_1 = (\lambda_\downarrow)_2$ Corollary 4 is trivial due to primal convergence. In the case $(\lambda_\downarrow)_1 > (\lambda_\downarrow)_2$, Corollary 4 follows due to local weak duality with an application of the main theorem.

One might conjecture that an analogous version to Corollary 4 is valid also for the third-highest entry and so on (meaning that the computed solution $x^{(T)}$ satisfies $(x_\downarrow^{(T)})_3 \leq (\lambda_\downarrow)_3 + \delta \lambda^*$ etc.). This is not the case in general. We provide a counterexample in the full version.

4 Limits of standard Dantzig-Wolfe-type algorithms

In this section, we study a class of algorithms that can be described by the meta-algorithm Algorithm 3. This class contains all algorithms that process the customers in a number T of phases and compute a solution $s_C^{(t)} \in X_C$ for every customer in each phase $t = 1, \dots, T$. We allow to choose any price vector for the oracle queries and allow to return any convex combination at the end of the algorithm. However, we fix the restricted step size rule in Line 7. On the one hand, this class can be interpreted as a natural generalization of the core algorithm. On the other hand, it includes standard conditional gradient methods that work with this restricted step size rule. Using this step-size rule is explained by the fact that otherwise $\|\xi b\|_\infty > 1$ holds. Since ξb is added to the current solution of the given phase and the goal is to find an element with small ℓ_∞ -norm, it is a reasonable approach to restrict the step-size towards elements with large ℓ_∞ -norm. Note that the bound $1/\|b\|_\infty$ in Line 7 can be replaced by $C/\|b\|_\infty$ for any other constant $C > 0$ without affecting the asymptotic lower bound. We prove that any such algorithm needs to use $\Omega(\frac{n+m}{\delta^2} \log m)$ many oracle calls to terminate with a $(1 + \delta)$ -approximate solution.

■ **Algorithm 3** Standard block-coordinate descent with restricted steps.

```

1: for  $t = 1, \dots, T$  do
2:   for  $C \in \mathcal{C}$  do
3:      $\alpha \leftarrow 0, s_C^{(t)} \leftarrow 0$ 
4:     while  $\alpha < 1$  do
5:       Choose  $y \in \mathbb{R}_{\geq 0}^m$ 
6:        $b \leftarrow f_C(y)$ 
7:        $\xi \leftarrow \min\{1 - \alpha, 1/\|b\|_\infty\}$ 
8:        $s_C^{(t)} \leftarrow s_C^{(t)} + \xi b$ 
9:        $\alpha \leftarrow \alpha + \xi$ 
10:    end while
11:   end for
12:    $s^{(t)} \leftarrow \sum_{C \in \mathcal{C}} s_C^{(t)}$ 
13: end for
14: return a convex combination of  $s^{(1)}, \dots, s^{(T)}$ 

```

► **Theorem 13.** *For every $\gamma \in (0, 1/2)$ there exist constants $K_\gamma, \tau_\gamma > 0$, such that for every $m > K_\gamma$, $n \in \mathbb{N}$, there exists an instance of the resource sharing problem with $n+2$ customers, $2m$ resources and $\lambda^* = 1$, such that for any $\delta \in (0, 1/10)$, any version of Algorithm 3 requires*

$$\tau_\gamma(n+m) \min \left\{ \frac{\log m}{\delta^2}, m^{1/2-\gamma} \right\} \quad (16)$$

oracle calls to compute a $(1+\delta)$ -approximate solution.

For a proof we refer to the full version.

5 Conclusion

In this work, we have presented an FPTAS for the primal and the dual of the resource sharing problem and improved on the best-known running time in terms of number of oracle calls.

We were able to show that our algorithm has the natural property to optimize (close to) independent parts of the instance separately by introducing the notion of local weak duality. This implied that our algorithm computes solutions that are close to decreasingly minimal on the two largest entries. Local weak duality provides a theoretical understanding of the empirically observed resilience to local effects of multiplicative-weight-based algorithms. Extending the algorithm to achieve (approximate) decreasing minimality on the third-highest entry and beyond is subject to future work.

In the last section, we have shown that further improvements, if possible, require different types of algorithms. A mere change of the price update rule is not enough to achieve faster convergence. This also implied that no warm-start analysis of any version of Algorithm 3 is possible.

References

- 1 Zeyuan Allen-Zhu and Lorenzo Orecchia. Nearly linear-time packing and covering lp solvers: Achieving width-independence and -convergence. *Mathematical Programming*, 175, February 2018. doi:10.1007/s10107-018-1244-x.

- 2 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. doi:10.4086/toc.2012.v008a006.
- 3 Daniel Bienstock and Garud Iyengar. Approximating fractional packings and coverings in $o(1/\epsilon)$ iterations. *SIAM J. Comput.*, 35:825–854, 2006.
- 4 Bhaskar Dutta and Debraj Ray. A concept of egalitarianism under participation constraints. *Econometrica*, 57:615–35, February 1989. doi:10.2307/1911055.
- 5 Lisa K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000. doi:10.1137/S0895480199355754.
- 6 András Frank and Kazuo Murota. Discrete decreasing minimization, part i: Base-polyhedra with applications in network optimization, 2019. arXiv:1808.07600.
- 7 András Frank and Kazuo Murota. Discrete decreasing minimization, part ii: Views from discrete convex analysis, 2020. arXiv:1808.08477.
- 8 András Frank and Kazuo Murota. Fair integral flows, 2020. arXiv:1907.02673.
- 9 Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956. doi:10.1002/nav.3800030109.
- 10 Satoru Fujishige. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Mathematics of Operations Research*, 5(2):186–196, 1980. URL: <http://www.jstor.org/stable/3689149>.
- 11 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37:630–652, January 2007. doi:10.1109/SFCS.1998.743463.
- 12 Michael Gester, Dirk Müller, Tim Nieberg, Christian Panten, Christian Schulte, and Jens Vygen. Bonnrout: Algorithms and data structures for fast and good vlsi routing. *ACM Trans. Des. Autom. Electron. Syst.*, 18(2), April 2013. doi:10.1145/2442087.2442103.
- 13 M. Grigoriadis and L. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM J. Optim.*, 4:86–107, 1994.
- 14 Michael D. Grigoriadis and Leonid G. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Mathematics of Operations Research*, 21(2):321–340, 1996. doi:10.1287/moor.21.2.321.
- 15 G. H. Hardy, George Polya, and J. E. Littlewood. *Inequalities*. The University press Cambridge [Eng.], 1934.
- 16 Stephan Held, Dirk Müller, Daniel Rotter, Rudolf Scheifele, Vera Traub, and Jens Vygen. Global routing with timing constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(2):406–419, 2018. doi:10.1109/TCAD.2017.2697964.
- 17 Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 427–435, Atlanta, Georgia, USA, 17–19 June 2013. PMLR. URL: <http://proceedings.mlr.press/v28/jaggi13.html>.
- 18 Klaus Jansen and Hu Zhang. Approximation algorithms for general packing problems and their application to the multicast congestion problem. *Mathematical Programming*, 114:183–206, 2008.
- 19 Thomas Kerdreux. *Accelerating conditional gradient methods*. PhD thesis, Université Paris sciences et lettres, June 2020.
- 20 Rohit Khandekar. *Lagrangian relaxation based algorithms for convex programming problems*. PhD thesis, Indian Institute of Technology Delhi, 2004.
- 21 Philip Klein and Neal E. Young. On the number of iterations for dantzig–wolfe optimization and packing-covering approximation algorithms. *SIAM Journal on Computing*, 44(4):1154–1172, 2015. doi:10.1137/12087222X.

- 22 Christos Koufogiannakis and Neal E. Young. A nearly linear-time ptas for explicit fractional packing and covering linear programs. *Algorithmica*, 70(4):648–674, 2014. Journal version of [2007]. doi:10.1007/s00453-013-9771-6.
- 23 Nimrod Megiddo. A good algorithm for lexicographically optimal flows in multi-terminal networks. *Bulletin of the American Mathematical Society*, 83:407–409, 1977.
- 24 Dirk Müller, Klaus Radke, and Jens Vygen. Faster min–max resource sharing in theory and practice. *Mathematical Programming Computation*, 3:1–35, 2011.
- 25 Dritan Nace and James Orlin. Lexicographically minimum and maximum load linear programming problems. *Operations Research*, 55:182–187, February 2007. doi:10.1287/opre.1060.0341.
- 26 Serge Plotkin, David Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20, January 2001. doi:10.1109/SFCS.1991.185411.
- 27 Bozidar Radunovic and Jean-Yves Le Boudec. A unified framework for max-min and min-max fairness with applications. *IEEE/ACM Transactions on Networking*, 15(5):1073–1083, 2007. doi:10.1109/TNET.2007.896231.
- 28 Prabhakar Raghavan and Clark D. Tompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, December 1987. doi:10.1007/BF02579324.
- 29 Farhad Shahrokhi and David W. Matula. The maximum concurrent flow problem. *J. ACM*, 37(2):318–334, April 1990. doi:10.1145/77600.77620.
- 30 Arie Tamir. Least majorized elements and generalized polymatroids. *Mathematics of Operations Research*, 20(3):583–589, 1995. doi:10.1287/moor.20.3.583.

On the External Validity of Average-Case Analyses of Graph Algorithms

Thomas Bläsius  

Karlsruhe Institute of Technology (KIT), Germany

Philipp Fischbeck  

Hasso Plattner Institute (HPI), University of Potsdam, Germany

Abstract

The number one criticism of average-case analysis is that we do not actually know the probability distribution of real-world inputs. Thus, analyzing an algorithm on some random model has no implications for practical performance. At its core, this criticism doubts the existence of *external validity*, i.e., it assumes that algorithmic behavior on the somewhat simple and clean models does not translate beyond the models to practical performance real-world input.

With this paper, we provide a first step towards studying the question of external validity systematically. To this end, we evaluate the performance of six graph algorithms on a collection of 2751 sparse real-world networks depending on two properties; the heterogeneity (variance in the degree distribution) and locality (tendency of edges to connect vertices that are already close). We compare this with the performance on generated networks with varying locality and heterogeneity. We find that the performance in the idealized setting of network models translates surprisingly well to real-world networks. Moreover, heterogeneity and locality appear to be the core properties impacting the performance of many graph algorithms.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Average Case, Network Models, Empirical Evaluation

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.21

Related Version *Full Version*: <https://arxiv.org/abs/2205.15066> [4]

Supplementary Material We provide our source code, a docker image for easier reproducibility, the real-world network data set as well as all generated data (networks and statistics):

Software (Source Code): <https://github.com/thobl/external-validity>

Dataset (Source Code Archive, Docker Image, Networks, Statistics): <https://doi.org/10.5281/zenodo.6587324>

1 Introduction

The seminal papers of Cook in 1971 [16] and Karp in 1972 [26] establish that many fundamental combinatorial problems are NP-hard, and thus cannot be solved in polynomial time unless $P = NP$. Since then, the list of NP-hard problems is growing every year.

Though the non-existence of polynomial time algorithms (unless $P = NP$) is major bad news, the concept of NP-hardness is limited to the worst case. It thus leaves the possibility of imperfect algorithms that fail sometimes but run correctly and in polynomial time on most inputs. Many algorithms used today are slow in the worst case but perform well on relevant instances. An early attempt to theoretically capture this “good in practice” concept is the *average-case analysis*. There, one assumes the input to be randomly generated and then proves a low failure probability or a good expected performance.¹ On that topic, Karp wrote in 1983 [27] that

¹ We note that within the scope of this paper the term *average case* refers to exactly those situations where the input is drawn from some probability distribution. This includes proving bounds that hold with high probability (instead of in expectation), which would technically be better described as *typical case*. Moreover, it excludes the case of randomized algorithms on deterministic inputs.



One way to validate or compare imperfect algorithms for NP-hard combinatorial problems is simply to run them on typical instances and see how often they fail. [...] While probabilistic assumptions are always open to question, the approach seems to have considerable explanatory power [...]. (Karp in 1983)

With this promising starting point, one could have guessed that average-case analysis is an important pillar of algorithmic research. However, it currently plays only a minor role in theoretical computer science. The core reason for this was summarized by Karp almost forty years later in 2020 in the Lex Fridman Podcast.²

The field tended to be rather lukewarm about accepting these results as meaningful because we were making such a simplistic assumption about the kinds of graphs that we would be dealing with. [...] After a while I concluded that it didn't have a lot of bite in terms of the practical application. (Karp in 2020)

At its core, this describes the issue that an average-case analysis is lacking *external validity*, i.e., the insights on randomly generated graphs do not transfer to practical instances.

The simplistic probabilistic assumption mentioned in the above quotes is that input graphs are drawn from the Erdős–Rényi model [23], where all edges exist independently at random with the same probability. This assumption has the advantages that it is as unbiased as possible and sufficiently accessible to allow for mathematical analyses. However, in its simplicity, it is unable to capture the rich structural properties present in real-world networks, leading to the lack of external validity.

That being said, since the beginnings of average-case considerations, there have been several decades of research in the field of network science dedicated to understanding and explaining properties observed in real-world networks. This in particular includes the analysis of random network models and the transfer of insights from these models to real networks; indicating external validity. Thus, we believe that it is time to revisit the question of whether average-case analyses of graph algorithms can have external validity. With this paper, we present a first attempt at systematically studying this question.

Before describing our approach and stating our contribution, we want to give two examples from network science, where the existence of external validity is generally accepted.

Examples from Network Science. The Barabási–Albert model [2] uses the mechanism of *preferential attachment* to iteratively build a network. Each newly added vertex chooses a fixed number of neighbors among the existing vertices with probabilities proportional to the current vertex degrees. This simple mechanism yields *heterogeneous* networks, i.e., networks with power-law degree distributions where most vertices have a small degree while few vertices have very high degree.³ It is well known that networks generated by this model are highly artificial, exhibiting properties that are far from what is observed in real-world networks. Nonetheless, beyond the specific model, it is generally accepted that the mechanism of preferential attachment facilitates power-law distributions. Thus, assuming external validity, the Barabási–Albert model can serve as an explanation of why we regularly observe power-law distributions in real-world data. Moreover, whenever we deal with a process involving preferential attachment, we should not be surprised when seeing a power-law distribution.

² Transcript of the Lex Fridman Podcast #111. The quote itself starts at 1:39:59. For the full context, start at 1:37:28 (<https://youtu.be/K11CrlfLuzs?t=5848>).

³ Barabási and Albert were not the first to study a preferential attachment mechanism; see, e.g., Price's model [19]. However, there is no doubt that their highly influential paper [2] popularized the concept.

The Watts–Strogatz model [32] first starts with a ring lattice, i.e., the vertices are distributed uniformly on a circle and vertex pairs are connected if they are sufficiently close. This yields a regular graph with high *locality*, i.e., all connections are short and we observe many triangles. Moreover, ring lattices have high diameter. The second step of the Watts–Strogatz model introduces noise by randomly rewiring edges. This diminishes locality by replacing local connections with potentially long-range edges. Watts and Strogatz demonstrate that only little locality has to be sacrificed before getting a small-world network with low diameter. Again, this model is highly artificial and thus far from being a good representation for real-world networks. However, it seems generally accepted that these observations have implications beyond the specific model, namely that there is a simple mechanism that facilitates the small-world property even in networks with mostly local connections. Thus, in real-world settings where random long-range connections are possible, one should not be surprised to observe the small-world property.

Contribution. We consider algorithms for six different problems that are known to perform better in practice than the worst-case complexity would suggest. We evaluate them on network models that allow for varying amounts of locality and heterogeneity.⁴ This shows us the impact of these two properties on the algorithms’ performance in the controlled and clean setting of generated networks. We compare this with practical performance by additionally evaluating the algorithms on a collection of 2751 sparse real-world networks. Our overall findings can be summarized as follows. Though the real-world networks yield a less clean and more noisy picture than the generated networks, the overall dependence of the performance on locality and heterogeneity coincides with surprising accuracy for most algorithms. This indicates that there is external validity in the sense that if, e.g., increasing locality in the network models improves performance, we should also expect better performance for real-world networks with high locality. Moreover, it indicates that locality and heterogeneity are the core properties to impact the performance for many networks.

Our insights for the specific algorithms are interesting in their own right, independent of the question of external validity. Moreover, our experiments led to several interesting findings that are beyond the core scope of this paper. These can be found in the full version [4].

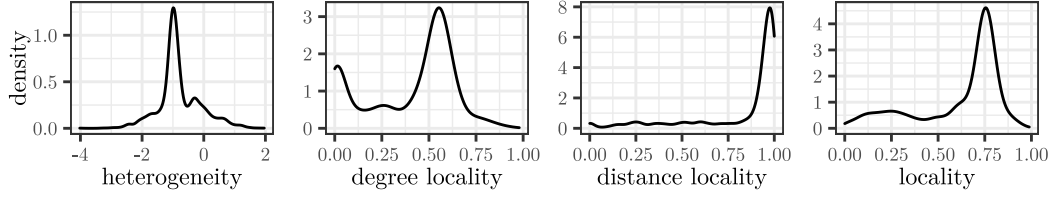
In Section 2, we introduce some basic notation and formally define measures for heterogeneity and locality. In Section 3, we describe the set of real-world and generated networks we use in our experiments. Section 4 compares generated and real-world networks for the different algorithms. Related work as well as our insights specific to the algorithms are discussed in this section. In Section 5 we conclude with a discussion of our overall results.

Our source code is available on GitHub⁵. It is additionally archived at Zenodo⁶, together with a docker image for easier reproducibility. The latter repository additionally includes the real-world network data set as well as all generated data (networks and statistics). Details omitted from this extended abstract can be found in the full version [4].

⁴ For non-local networks, we use the Erdős–Rényi and the Chung–Lu model for homogeneous and heterogeneous degree distributions, respectively. For local networks, we use geometric inhomogeneous random graphs (GIRGs), which let us vary the amount of locality and heterogeneity.

⁵ <https://github.com/thobl/external-validity>

⁶ <https://doi.org/10.5281/zenodo.6587324>



■ **Figure 1** The density (kernel density estimation) of heterogeneity, degree locality, distance locality, and locality of the networks in our data set of real-world networks.

2 Basic Definitions, Heterogeneity, and Locality

Let $G = (V, E)$ be a graph. Throughout the paper, We denote the number of vertices and edges with $n = |V|$ and $m = |E|$. For $v \in V$, let $N(v) = \{u \mid \{u, v\} \in E\}$ be the *neighborhood* of v , and let $\deg(v) = |N(v)|$ be the *degree* of v . Additionally, $N[v] = N(v) \cup \{v\}$ is the *closed neighborhood* of v . An edge $e \in E$ is a *bridge* if $G - e$ is disconnected, where $G - e$ denotes the subgraph induced by $E \setminus \{e\}$.

We define the *heterogeneity* of a graph as the logarithm (base 10) of the coefficient of variation of its degree distribution, i.e., it is $\log_{10}(\sigma/\mu)$ for the average degree $\mu = \frac{1}{n} \sum_{v \in V} \deg(v)$ and the variance $\sigma^2 = \frac{1}{n} \sum_{v \in V} (\deg(v) - \mu)^2$.

We define the locality based on two different measures. For $u, v \in V$, let $\deg(u, v) = |N(u) \cap N(v)|$ be the *common degree* of u and v . For a non-bridge edge $\{u, v\} \in E$ the *degree locality* is defined as

$$L_{\deg}(\{u, v\}) = \frac{\deg(u, v)}{\min(\deg(u), \deg(v)) - 1}.$$

It essentially measures in how many triangles $\{u, v\}$ appears. The *degree locality* $L_{\deg}(G)$ of G is the average degree locality over all non-bridge edges.

For $u, v \in V$, let $\text{dist}(u, v)$ be the distance between u and v in G . For $P \subseteq \binom{V}{2}$, we denote the average distance between vertex pairs in P with $\text{dist}(P)$. For $\{u, v\} \in E$, the *detour distance* $\text{dist}^+(u, v)$ is the distance between u and v in $G - \{u, v\}$. Let $\bar{E} = \binom{V}{2} \setminus E$ and assume that $\bar{E} \neq \emptyset$. For a non-bridge edge $\{u, v\} \in E$, define the *distance locality* as

$$L_{\text{dist}}(\{u, v\}) = 1 - \frac{\text{dist}^+(u, v) - 2}{\text{dist}(\bar{E}) - 2}.$$

It essentially measures how short the edge $\{u, v\}$ is compared to the average distance in the graph. If $\text{dist}(\bar{E}) = 2$, we define $L_{\text{dist}}(\{u, v\}) = 0$. The *distance locality* $L_{\text{dist}}(G)$ of G is the maximum of 0 and the average distance locality over all non-bridge edges.

The *locality* of G is $L(G) = (L_{\deg}(G) + L_{\text{dist}}(G))/2$. To compute the locality, we approximate average distances [12]. See Figure 1 for distribution plots.

3 Networks

Real-World Networks. We use 2751 graphs from Network Repository [29], which we selected as follows. We started with all networks with at most 1 M edges and reduced each network to its largest connected component. We removed multi-edges and self-loops and ignored weights or edge directions. We tested the resulting connected, simple, undirected, and unweighted graphs for isomorphism and kept only one copy for each group of isomorphic networks. This resulted in 3006 networks. From this we removed 255 networks for different reasons.

- We removed 105 networks that were randomly generated and thus do not count as real-world networks.⁷
- We removed 17 trees. They are not very interesting, and locality is not defined for trees.
- We removed 133 graphs with density at least 10%. Our focus lies on sparse graphs and network models for sparse graphs. Thus, dense graphs are out of scope.

Random Networks. We use three random graph models to generate networks; the Erdős–Rényi model [23] (non-local and homogeneous), the Chung–Lu model [14, 15] (non-local and varying heterogeneity), and the GIRG model [11] (varying locality and heterogeneity). For the latter, we use the efficient implementation in [8].

For each model and parameter configuration, we generate five networks with $n = 50\text{ k}$ vertices and (expected) average degree 10. As for the real-world networks, we reduce each generated graph to its largest connected component. We average the five generated graphs for each parameter configuration and represent their average as single dot in the plots.

4 Comparison Between the Models and Real-World Networks

Each of the following subsections compares the performance of a different algorithm between generated and real-world networks. For the cost c of the algorithm, we plot c depending on heterogeneity and locality using color to indicate the cost; see, e.g., Figure 2. The left and middle plot show one data point for each parameter setting of the models and each real-world network, respectively. The right plot aggregates real-world networks with similar locality and heterogeneity. Each point represents a number of networks indicated by its radius (log scale). We regularly assume the cost c to be polynomial in m (or n), i.e., $c = m^x$, and plot $x = \log_m c$. In this case, the color in the left and right plot shows the mean exponent x for the aggregated networks.

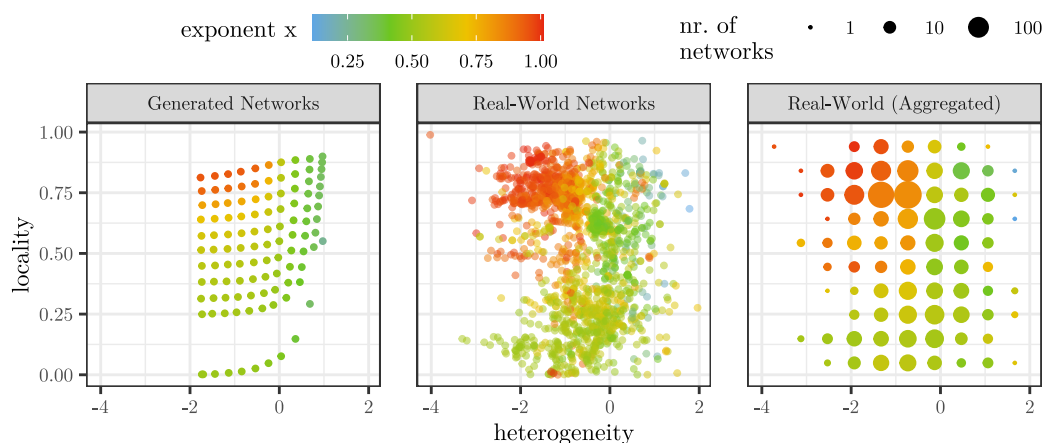
4.1 Bidirectional Search

We can compute a shortest path between two vertices s and t using a *breadth first search* (*BFS*). The BFS explores the graph layer-wise, where the i th layer contains the vertices of distance i from s . By *exploring* the i th layer, we refer to the process of iterating over all edges with an endpoint in the i th layer, thereby finding layer $i + 1$. The *bidirectional BFS* alternates the exploration of layers between a *forward BFS* from s and a *backward BFS* from t . The alternation strategy we study here greedily applies the cheaper of the two explorations in every step. The cost of exploring a layer is estimated via the sum of degrees of vertices in that layer. The search stops once the current layers of forward and backward search intersect.

The *cost* c for the bidirectional BFS is the average number of edge explorations over 100 random st -pairs. Note that $c \leq 2m$, as each edge can be explored at most twice; once from each side. Figure 2 shows the exponent x for $c = m^x$ depending on heterogeneity and locality.

Impact of Locality and Heterogeneity. For the generated networks, we see that networks with high locality and homogeneous degree distribution (top left corner) have an exponent of around 1 (red). Thus, the cost of the bidirectional search is roughly m , which matches

⁷ Finding generated networks was done manually by searching for suspicious naming patterns or graph properties. For each candidate, we checked its source to verify that it is a generated network. Though we checked thoroughly, there are probably a few random networks hiding among the real-world networks.



■ **Figure 2** The exponent x of the average cost $c = m^x$ of the bidirectional BFS over 100 st -pairs.

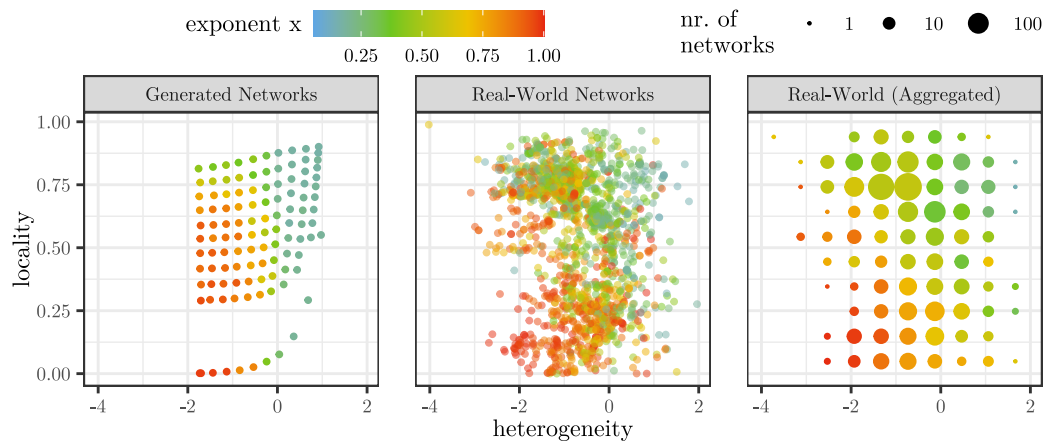
the worst-case bound. If the network is heterogeneous (right) or less local (bottom), we get significantly lower exponents of around 0.5, indicating a cost of roughly \sqrt{m} . The cost is particularly low for very heterogeneous networks. Overall, we get a strict separation between hard (high locality, low heterogeneity) and easy (low locality or high heterogeneity) instances.

For real-world networks, we observe the same overall behavior that instances that are local and homogeneous tend to be hard while all others tend to be easy. There are only few exceptions to this, indicating that the heterogeneity and locality are usually the crucial properties impacting the performance of the bidirectional BFS.

Discussion. Borassi and Natale [10] found that the bidirectional BFS is exceptionally efficient on many real-world networks. Though its efficiency is surprising when compared to the worst case, existing average-case considerations suggest that this is actually the expected behavior. It runs in sublinear time on the non-local Erdős–Rényi and Chung–Lu graphs [10]. It is also sublinear on the local and heterogeneous hyperbolic random graphs, but linear on geometric random graphs in Euclidean geometry [7]. The latter two models can be seen as special case of GIRGs, covering the top-right and top-left corner.

4.2 Diameter

The *eccentricity* of $s \in V$ is $\max_{t \in V} \text{dist}(s, t)$, i.e., the distance to the vertex farthest from s . The *diameter* of the graph G is the maximum eccentricity of its vertices. It can be computed using the *iFUB algorithm* [17]. It starts with a root $r \in V$ from which it computes the BFS tree T . It then processes the vertices bottom up in T and computes their eccentricities using a BFS per vertex. This process can be pruned when the distance to r is sufficiently small compared to the largest eccentricity found so far. Pruning works well if r is a central vertex in the sense that it has low distance to many vertices and a shortest path between distant vertices gets close to r . The central vertex r is selected as follows. A *double sweep* [28] starts with a vertex u , chooses a vertex v at maximum distance from u , and returns a vertex w from a middle layer of the BFS tree from v . A *4-sweep* [17] consists of two double sweeps, starting the second sweep with the result w of the first sweep. We consider the *iFUB+4-sweep* algorithm, which chooses r by doing a 4-sweep from a vertex of maximum degree.



■ **Figure 3** The exponent x of the number of BFSs $c = n^x$ of the iFUB+4-sweepshd algorithm, excluding 19 real-world networks with timeout. The GIRG ground space is a square.

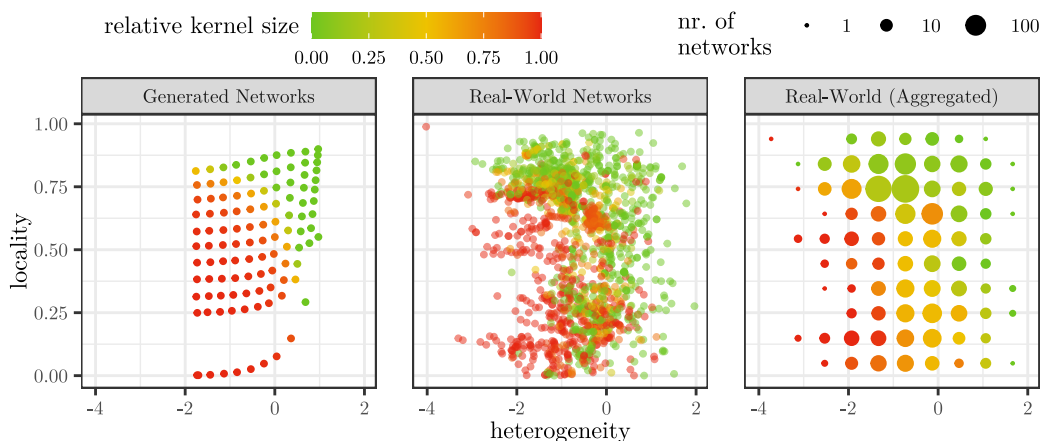
The *cost* c of iFUB+4-sweepshd is the number of BFSs it performs. Note that $c \leq n$. Figure 3 shows the exponent x for $c = n^x$ depending on heterogeneity and locality. It excludes 19 of the 2751 real-world networks that exceeded the time limit of 30 min. The GIRG model uses a square as ground space instead of the usual torus.

Impact of Locality and Heterogeneity. The general dependence is the same for generated and real-world networks. An almost linear number of BFSs is required for networks lacking locality and heterogeneity. For local or heterogeneous networks, it is substantially sublinear. The picture for real-world networks shows some noise, which indicates that there are properties besides locality and heterogeneity that impact the performance. We note that this observation agrees with the models, where we get highly varying exponents for individual parameter settings, e.g., for GIRGs with $T = 0.82$ and $\beta = 3.4$, we get exponents ranging from 0.21 to 0.92 for the five generated instances.

Discussion. The iFUB algorithm was introduced by Crescenzi, Grossi, Habib, LANZI, and Marino [17]. They note that it works well on networks with high difference between radius and diameter. This corresponds to the existence of a central vertex with eccentricity close to half the diameter. Additional experiments indicate that locality facilitates the existence of a central vertex (essentially in the center of the geometric ground space). Moreover, in heterogeneous networks, the high degree vertices serve as central vertices. The latter is also supported by the theoretical analysis of Borassi, Crescenzi, and Trevisan [9], who in particular showed that heterogeneous Chung–Lu graphs allow the computation of the diameter in sub-quadratic time, indeed using a vertex of high degree as central vertex.

4.3 Vertex Cover Domination

A vertex set $S \subseteq V$ is a *vertex cover* if every edge has an endpoint in S , i.e., removing S from G leaves a set of isolated vertices. We are interested in finding a vertex cover of minimum size. For two adjacent vertices $u, v \in V$, we say that u *dominates* v if $N[v] \subseteq N[u]$. The *dominance rule* states that there exists a minimum vertex cover that includes u . Thus, one can reduce the instance by including u in the vertex cover and removing it from the graph.



■ **Figure 4** The relative kernel size of the vertex cover domination rule.

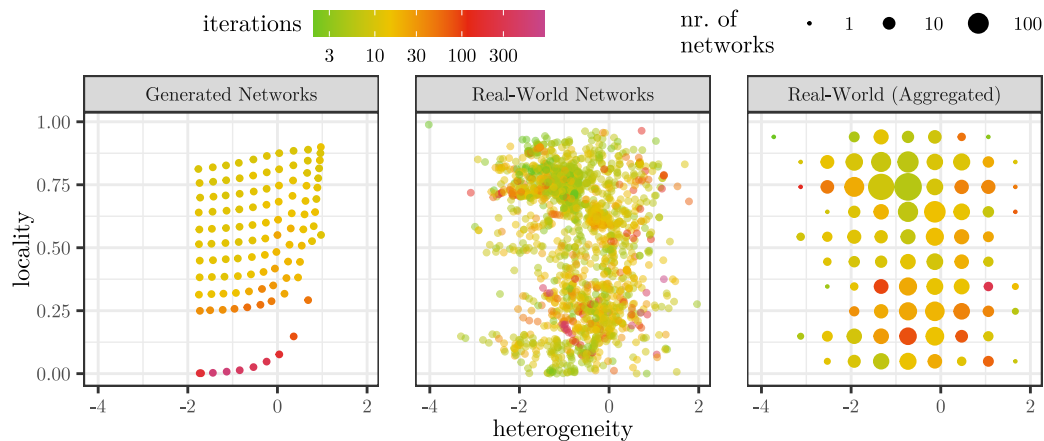
To evaluate the effectiveness of the dominance rule, we apply it exhaustively, i.e., until no dominant vertices are left. Moreover, we remove isolated vertices. We refer to the number c of vertices in the largest connected component of the remaining instances as the *kernel size*. Figure 4 shows the *relative kernel size* c/n with respect to locality and heterogeneity.

Impact of Locality and Heterogeneity. We see a sharp separation for the generated networks. For low locality and heterogeneity (bottom left), the reduction rule cannot be applied. For high locality and heterogeneity (top right), the dominance rule completely solves the instance. For real-world networks, the separation is less sharp, i.e., there is a larger range of locality/heterogeneity values in the middle where the dominance rule is effective sometimes. Nonetheless, we see the same trend that the reduction rule is more likely to be effective the higher the locality and heterogeneity. In the extreme regimes (bottom left or top right), we observe the same behavior as for the generated networks with relative kernel sizes close to 1 and 0, respectively, for almost all networks. Moreover, there is dichotomy in the sense that many instances are either (almost) completely solved by the dominance rule or the rule is basically inapplicable.

Discussion. Though vertex cover is NP-hard [26], it is rather approachable: It can be solved in $1.1996^n n^{O(1)}$ time [34] and there is a multitude of FPT-algorithms with respect to the solution size k [18], the fastest one running in $O(1.2738^k + kn)$ [13]. Moreover, there are good practical algorithms [1, 33]. Both algorithms apply a suite of reduction rules, including the dominance rule or a generalization. We note that the dominance rule is closely related to Weihe’s reduction rules for hitting set [33]. Previous experiments for Weihe’s reduction rules match our results: they work well if the instances are local and heterogeneous [6]. Concerning theoretic analysis on models, we know that on hyperbolic random graphs, the dominance rule is sufficiently effective to yield a polynomial time algorithm [5]. Thus, it is not surprising that the top-right corner in Figure 4 is mostly green.

4.4 The Louvain Algorithm for Graph Clustering

Let $V_1 \cup \dots \cup V_k = V$ be a *clustering* where each vertex set V_i is a *cluster*. One is usually interested in finding clusterings with dense clusters and few edges between clusters, which is formalized using some quality measure, e.g., the so-called modularity. A common subroutine



■ **Figure 5** Number of iterations of the first local search of the Louvain algorithm. The color scale is logarithmic. Four outliers (real-world) with more than 1 k iterations are excluded.

in clustering algorithms is to apply the following local search. Start with every vertex in its own cluster. Then, check for each vertex $v \in V$ whether moving v into a neighboring cluster improves the clustering. If so, v is moved into the cluster yielding the biggest improvement. This is iterated until no improvement can be achieved. Doing this with the modularity as quality measure (and subsequently repeating it after contracting clusters) yields the well-known Louvain algorithm [3].

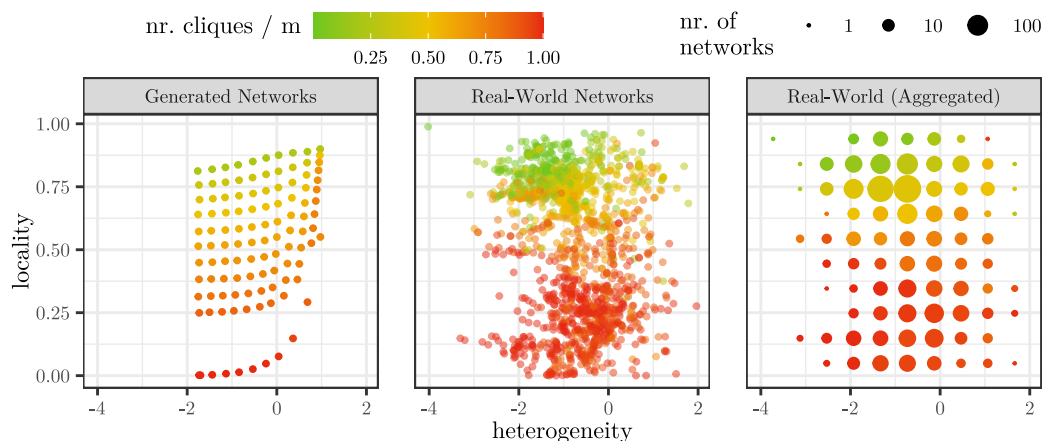
The run time of the Louvain algorithm is dominated by the number of iterations of the initial local search. Figure 5 shows this number of iterations.

Impact of Locality and Heterogeneity. For the generated instances, the number of iterations is generally low but increasing when decreasing the locality. For GIRGs, the average number of iterations ranges from 10.2 to 54.2 for the different parameter configurations. Moreover, the number of iterations starts to rise only for low localities. For 86 % of the configurations the average number of iterations lies below 20. For the Erdős–Rényi graphs we obtain an average of 196.4 iterations and for Chung–Lu graphs it goes up to 591.4 for one configuration. However, these average values over five generated instances have to be taken with a grain of salt as there is a rather high variance, e.g., the number of iterations for the five Chung–Lu graphs with power-law exponent 25 ranges from 82 up to 312.

For the real-world networks there is no clear trend depending on locality or heterogeneity. In general, the number of iterations is rather low except for some outliers. While the strongest outlier requires almost 20 k iterations, 98.6 % of the networks have at most 100 iterations.

Discussion. The worst-case number of iterations of the Louvain algorithm can be upper bounded by $O(m^2)$ due to the fact that the modularity is bounded and each vertex move improves it by at least $\Omega(1/m^2)$. Moreover, there exists a graph family that requires $\Omega(n)$ iterations for the first local search [25, Proposition 3.1].

Our experiments indicate that locality or heterogeneity are not the properties that discriminate between easy and hard instances. For generated instances, there is the trend that low locality increases the number of iterations, which does not transfer to the real-world networks (or is at least less clearly). However, the general picture that most instances require few iterations while there are some outliers coincides for generated and real-world networks.



■ **Figure 6** The number of maximal cliques relative to m depending on the heterogeneity and locality, restricted to networks where this value is at most 1 (93% of the networks).

4.5 Maximal Cliques

A *clique* is a subset of vertices $C \subseteq V$ such that C induces a complete graph, i.e., any pair of vertices in C is connected. A clique is *maximal*, if it is not contained in any other clique. In the following, we are never interested in non-maximal cliques. Thus, whenever we use the term *clique*, it implicitly refers to a maximal clique. To enumerate all cliques, we used the algorithm by Eppstein, Löffler, and Strash [20], using their implementation [21, 22].

As the cliques of a network can be enumerated in polynomial time per clique [30], the number of maximal cliques is a good indicator for the hardness of an instance. For all generated and most real-world networks, the number of cliques does not exceed the number of edges m . Out of the 2751 real-world networks, 2556 (93%) have at most m and 193 have more than m cliques. For the remaining 2 networks, the timeout of 30 min was exceeded. Figure 6 shows the number of cliques relative to m for all networks with at most m cliques.

Impact of Locality and Heterogeneity. One can see that the networks (generated and real-world) with low locality have roughly m cliques, while the number of cliques decreases for increasing locality. Moreover, among networks with locality, there is the slight trend that networks with higher heterogeneity have more cliques.

It is not surprising that networks with low locality have roughly m cliques, as graphs without triangles have exactly m cliques. For graphs with higher locality, there are two effects counteracting each other. On the one hand, multiple edges combine into larger cliques, which decreases the number of cliques. On the other hand, each edge can be contained in multiple cliques, which increases the number of cliques. Our experiments show that the former effect usually supersedes the latter, i.e., the size of the cliques increases more than the number of cliques each edge is contained in.

Discussion. There are many results on enumerating cliques and on the complementary problem of enumerating independent sets. Here, we focus on discussing two results that are closely related. Eppstein, Löffler and Strash [21] give an algorithm for enumerating all cliques that runs in $O(dn3^{d/3})$ time, where d is the degeneracy (a measure for sparsity). We use this algorithm for our experiments as it performs well in practice.

Fox, Roughgarden, Seshadhri, Wei, and Wein [24] introduced the closure as a measure that captures the tendency that common neighbors facilitate direct connections, i.e., as a measure for locality. Additionally, they introduced the weak closure as a more robust measure. Fox et al. show that weakly c -closed graphs have at most $3^{(c-1)/3}n^2$ cliques. For c -closed graphs they give the additional upper bound of $4^{(c+4)(c-1)/2}n^{2-2^{1-c}}$.

Qualitatively, at a first glance, these theoretical results seem to match our observations on real-world networks: sparse graphs contain few cliques and there are fewer cliques for more local networks (small c). However, a closer look reveals two caveats. First, both upper bounds are exponential in the parameter while 93% of real-world networks have at most m cliques. Second, the (weak) closure does not actually correlate with the number of cliques or the locality. Weak closure and degeneracy only seem to be a good measure for hardness on the remaining 7% of the instances. Thus, on the large majority of instances, the generative models yield a much better fit.

4.6 Chromatic Number

We study the effectiveness of a reduction rule for computing the chromatic number based on the size k of the maximum clique [31]. It reduces a graph to its k -core and is thus closely related to the degeneracy of the network.

Whether the reduction rule performs well comes, at its core, down to how the clique size k compares to the degeneracy d . We observe that both, k and d , mainly depend on locality and heterogeneity and behave very similar on real-world and generated networks. However, k and d have the same dependence on locality and heterogeneity, which means that (at least for some regimes) other properties have to tip the scale. We in particular observe a dependence on the average degree for the generated networks; the higher the average degree, the worse the reduction rule performs. More details on this can be found in the full version of the paper [4].

5 Discussion and Conclusion

Networks from different domains have varying structural properties. Thus, trying to find probability distributions that match or closely approximate those of real-world networks seems like a hopeless endeavor. Moreover, even if we knew such a probability distribution, it would most likely be highly domain specific and too complicated for theoretical analysis.

Our View on Average-Case Analysis. A more suitable approach to average case analysis is the use of models that assume few specific structural properties and are as unbiased as possible beyond that. If the chosen properties are the dominant factors for the algorithm's performance, we obtain external validity, i.e., the results translate to real-world instances even though they do not actually follow the assumed probability distributions. There are two levels of external validity.

- The models capture the performance-relevant properties sufficiently well that algorithms perform similar on generated and real-world networks.
- The models are too much of an idealization for this direct translation to practical performance. However, varying a certain structural property in this idealized world has the same qualitative effect on performance as it has on real-world networks.

Though an average case analysis cannot yield strong performance guarantees, with the above notions of external validity, it can give insights into what properties are crucial for performance (first level) and how the performance depends on a property (second level). Moreover, even a

lack of external validity can yield valuable insights in the following sense. Assume we have the hypothesis that property X is the crucial factor for algorithmic performance and thus we study a model with property X as null model. Then, a lack of external validity lets us refute the hypothesis as there clearly has to be a different property impacting the performance.

Impact of Locality and Heterogeneity. Non-surprisingly, the performance on real-world networks depending on locality and heterogeneity is more noisy compared to the generated networks, as real-world networks are diverse and vary in more than just these two properties. That being said, the observations on the models and in practice coincide almost perfectly for the bidirectional search and the enumeration of maximal cliques. For the maximal cliques, the match even includes constant factors, which is particularly surprising, as these numbers are below m while the worst case is exponential.

For the vertex cover domination rule as well as the iFUB algorithm, we obtain a slightly noisier picture. However, the overall trend matches well, which indicates that locality and heterogeneity are crucial factors for the performance, but not the only ones. For the iFUB algorithm, we already identified the existence of central vertices as additional factor (difference between torus and square as underlying geometry).

For the chromatic number, we observe that heterogeneity and locality are important but not the only factors impacting performance. Nonetheless, the performance is similar on the models compared to real-world networks.

For the Louvain clustering algorithm, the models and real-world networks coincide insofar, that the number of iterations is low, with few exceptions. This indicates that locality and heterogeneity are not the core properties for differentiating between hard and easy instances.

Conclusions. Locality and heterogeneity have significant impact on the performance of many algorithms. We believe that it is useful for the design of efficient algorithms to have these two dimensions of instance variability in mind. Moreover, GIRGs [11] with the available efficient generator [8] provide an abundance of benchmark instances on networks with varying locality and heterogeneity. Finally, we believe that average case analyses on the four extreme cases can help to theoretically underpin practical run times. The four extreme cases can, e.g., be modeled using geometric random graphs for local plus homogeneous, hyperbolic random graphs⁸ for local plus heterogeneous, Erdős–Rényi graphs for non-local plus homogeneous, and Chung–Lu graphs for non-local plus heterogeneous networks.

References

- 1 Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.
- 2 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. doi:10.1126/science.286.5439.509.
- 3 Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008. doi:10.1088/1742-5468/2008/10/p10008.
- 4 Thomas Bläsius and Philipp Fischbeck. On the external validity of average-case analyses of graph algorithms. *Computing Research Repository (CoRR)*, abs/2205.15066, 2022. doi:10.48550/arXiv.2205.15066.

⁸ GIRGs can be seen as common generalization of geometric and hyperbolic random graphs.



- 5 Thomas Bläsius, Philipp Fischbeck, Tobias Friedrich, and Maximilian Katzmann. Solving vertex cover in polynomial time on hyperbolic random graphs. *Theory of Computing Systems*, 2021. doi:10.1007/s00224-021-10062-9.
- 6 Thomas Bläsius, Philipp Fischbeck, Tobias Friedrich, and Martin Schirneck. Understanding the effectiveness of data reduction in public transportation networks. In *Workshop on Algorithms and Models for the Web-Graph (WAW)*, pages 87–101, 2019. doi:10.1007/978-3-030-25070-6_7.
- 7 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. *ACM Transactions on Algorithms (TALG)*, 18(2):19:1–19:32, 2022. doi:10.1145/3516483.
- 8 Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck, and Christopher Weyand. Efficiently generating geometric inhomogeneous and hyperbolic random graphs. In *European Symposium on Algorithms (ESA)*, pages 21:1–21:14, 2019. doi:10.4230/LIPIcs.ESA.2019.21.
- 9 Michele Borassi, Pierluigi Crescenzi, and Luca Trevisan. An axiomatic and an average-case analysis of algorithms and heuristics for metric properties of graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 920–939, 2017. doi:10.1137/1.9781611974782.58.
- 10 Michele Borassi and Emanuele Natale. KADABRA is an ADaptive algorithm for betweenness via random approximation. *ACM Journal of Experimental Algorithmics (JEA)*, 24(1):1.2:1–1.2:35, 2019. doi:10.1145/3284359.
- 11 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.
- 12 Shiri Chechik, Edith Cohen, and Haim Kaplan. Average distance queries through weighted samples in graphs and metric spaces: High scalability with tight statistical guarantees. In *International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 659–679, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.659.
- 13 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.
- 14 Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882, 2002. doi:10.1073/pnas.252631999.
- 15 Fan Chung and Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6(2):125–145, 2002. doi:10.1007/PL00012580.
- 16 Stephen A. Cook. The complexity of theorem-proving procedures. In *Symposium on the Theory of Computing (STOC)*, pages 151–158, 1971. doi:10.1145/800157.805047.
- 17 Pilu Crescenzi, Roberto Grossi, Michel Habib, Leonardo LANZI, and Andrea Marino. On computing the diameter of real-world undirected graphs. *Theoretical Computer Science*, 514:84–95, 2013. doi:10.1016/j.tcs.2012.09.018.
- 18 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.
- 19 Derek J. de Solla Price. A general theory of bibliometric and other cumulative advantage processes. *Journal of the Association for Information Science and Technology (JASIST)*, 27(5):292–306, 1976. doi:10.1002/asi.4630270505.
- 20 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 403–414, 2010. doi:10.1007/978-3-642-17517-6_36.
- 21 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics (JEA)*, 18, 2013. doi:10.1145/2543629.

- 22 David Eppstein and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. In *Symposium on Experimental and Efficient Algorithms (SEA)*, pages 364–375, 2011. doi:10.1007/978-3-642-20662-7_31.
- 23 P. Erdős and A. Rényi. On random graphs I. *Publicationes Mathematicae*, 6:290–297, 1959. URL: https://www.renyi.hu/~p_erdos/1959-11.pdf.
- 24 Jacob Fox, Tim Roughgarden, C. Seshadhri, Fan Wei, and Nicole Wein. Finding cliques in social networks: A new distribution-free model. *SIAM Journal on Computing*, 49(2):448–464, 2020. doi:10.1137/18M1210459.
- 25 Andrea Kappes. *Engineering Graph Clustering Algorithms*. PhD thesis, Karlsruhe Institute of Technology, 2015. URL: <http://digbib.ubka.uni-karlsruhe.de/volltexte/1000049269>.
- 26 Richard M. Karp. Reducibility among combinatorial problems. In *Computational Complexity Conference (CCC)*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 27 Richard M. Karp. The probabilistic analysis of combinatorial optimization algorithms. In *International Congress of Mathematicians (ICM)*, volume 2, pages 1601–1609, 1983. URL: <https://www.mathunion.org/fileadmin/ICM/Proceedings/ICM1983.2/ICM1983.2.ocr.pdf>.
- 28 Clémence Magnien, Matthieu Latapy, and Michel Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *ACM Journal of Experimental Algorithmics (JEA)*, 13, 2008. doi:10.1145/1412228.1455266.
- 29 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 4292–4293, 2015. URL: <http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9553>.
- 30 Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977. doi:10.1137/0206036.
- 31 Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015. doi:10.1287/ijoc.2014.0618.
- 32 Duncan J. Watts and Steven H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, 1998. doi:10.1038/30918.
- 33 Karsten Weihe. Covering trains by stations or the power of data reduction. In *Algorithms and Experiments (ALEX)*, pages 1–8, 1998.
- 34 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.

On Polynomial Kernels for Traveling Salesperson Problem and Its Generalizations

Václav Blažej  

Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic

Pratibha Choudhary  

Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic

Dušan Knop  

Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic

Šimon Schierreich  

Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic

Ondřej Suchý  

Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic

Tomáš Valla  

Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic

Abstract

For many problems, the important instances from practice possess certain structure that one should reflect in the design of specific algorithms. As data reduction is an important and inextricable part of today's computation, we employ one of the most successful models of such precomputation – the kernelization. Within this framework, we focus on TRAVELING SALESPERSON PROBLEM (TSP) and some of its generalizations.

We provide a kernel for TSP with size polynomial in either the feedback edge set number or the size of a modulator to constant-sized components. For its generalizations, we also consider other structural parameters such as the vertex cover number and the size of a modulator to constant-sized paths. We complement our results from the negative side by showing that the existence of a polynomial-sized kernel with respect to the fractioning number, the combined parameter maximum degree and treewidth, and, in the case of SUBSET TSP, modulator to disjoint cycles (i.e., the treewidth two graphs) is unlikely.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases Traveling Salesperson, Subset TSP, Waypoint Routing, Kernelization

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.22

Related Version *Full Version:* <https://arxiv.org/abs/2207.01109>

Funding The authors acknowledge the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 “Research Center for Informatics”. This work was additionally supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS20/208/OHK3/3T/18.



© Václav Blažej, Pratibha Choudhary, Dušan Knop, Šimon Schierreich, Ondřej Suchý, and Tomáš Valla; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 22; pp. 22:1–22:16

Leibniz International Proceedings in Informatics



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

1 Introduction

The TRAVELING SALESPERSON PROBLEM is among the most popular and most intensively studied combinatorial optimization problems in graph theory¹. From the practical point of view, the problem was already studied in the 1950s; see, e.g., [15]. We define the problem formally as follows.

TRAVELING SALESPERSON PROBLEM (TSP)

Input: An undirected graph $G = (V, E)$, edge weights $\omega: E \rightarrow \mathbb{N}$, and a budget $b \in \mathbb{N}$.

Question: Is there a closed walk in G of total weight at most b that traverses each vertex of G at least once?

TSP is known to be NP-hard [24]. It is worth mentioning that the input to TSP is usually a complete graph and it is required that each vertex is visited exactly once (single-visit version). In this paper, we aim to study the impact of the structure of the input graph on the complexity of finding a solution to TSP. Towards this, we employ parameterized analysis [16] and therefore we also consider the decision version of TSP. Many variants of the problem have been studied in detail [29]. Our formulation is also known as GRAPHICAL TSP [14].

In this paper, we further study two natural generalizations of TSP – SUBSET TSP and WAYPOINT ROUTING PROBLEM. In SUBSET TSP (SUBTSP) the objective is to find a closed walk required to traverse only a given subset W of vertices (referred to as *waypoints*) instead of the whole vertex set as is in TSP. An instance of TSP can be interpreted as an instance of SUBTSP by letting $W = V(G)$. In WAYPOINT ROUTING PROBLEM (WRP) the edges of the input graph have capacities (given by a function $\kappa: E(G) \rightarrow \mathbb{N}$) and the objective is to find a closed walk traversing a given subset of vertices respecting the edge capacities, i.e., there is an upper limit on the number of times each edge can be traversed in a solution. As we show later, an instance of SUBTSP can be interpreted as an instance of WRP by letting $\kappa(e) = 2$ for every $e \in E(G)$. Note that both SUBTSP and WRP are NP-complete.

Related Work. TSP (and its variants) were extensively studied from the viewpoint of approximation algorithms. The single-visit version of TSP in general cannot be approximated, unless $P = NP$ [51]. For the metric single-visit version of the problem Christofides [13] provided a $\frac{3}{2}$ -approximation, while it is known that unless $P = NP$, there is no $\frac{117}{116}$ -approximation algorithm [12]. This is so far the best known approximation algorithm for the general case, despite a considerable effort, e.g., [54, 26, 10, 23, 8, 30, 37].

The problem remains APX-hard even in the case of weights 1 and 2, however, a $\frac{7}{6}$ -approximation algorithm is known for this special case [50]. A PTAS is known for the special case of Euclidean [4, 47] and planar [27, 3, 38] TSP. Also, the case of graph metrics received significant attention. Gharan et al. [25] found a $\frac{3}{2} - \epsilon_0$ approximation for this variant. Mömke and Svensson [48] then obtained a combinatorial algorithm for graphic TSP with an approximation factor of 1.461. This was later improved by Mucha [49] to $\frac{13}{9}$ and then by Sebö and Vygen [53] to 1.4. See, e.g., the monograph of Applegate et al. [2] for further information.

A popular practical approach to the single-visit version of TSP is to gradually improve some given solution using local improvements – the so-called q -OPT moves (see, e.g., [35, 36]). Marx [45] proved that TSP is W[1]-hard w.r.t. q . Later, Bonnet et al. [7] studied the q -OPT

¹ TSP Homepage: <http://www.math.uwaterloo.ca/tsp/index.html>.

technique on bounded degree graphs. They also investigated the complexity in the case where q is a fixed constant. The first significant theoretical improvement in the general case was by de Berg et al. [6] (announced at ICALP '16) who gave an $\mathcal{O}(n^{\lfloor 2q/3 \rfloor + 1})$ time dynamic programming based algorithm. Furthermore, they showed that improving upon the $\mathcal{O}(n^3)$ time for the case of 3-OPT would yield a breakthrough result by a connection with the ALL PAIRS SHORTEST PATHS (APSP) problem (see, e.g., [11]). Lancia and Dalpasso [40] observed that the DP-based approach is indeed slow in practice and implemented a practical fast algorithm tailored for 4-OPT. Later, Cygan et al. [17] improved the result of de Berg et al. [6] by presenting an $\mathcal{O}(n^{(1/4+\varepsilon_q)q})$ time algorithm. Moreover, they showed a connection of 4-OPT to APSP. Guo et al. [28] considered other suitable measures (parameters) for local search (e.g., r -swap or r -edit) and provided a W[1]-hardness result in general graphs (for both parameters). Furthermore, they gave an FPT algorithm for planar graphs and showed that the existence of polynomial kernels is unlikely even in this case.

SUBSET TSP can be solved in time $2^\ell \cdot n^{\mathcal{O}(1)}$ by first computing the distance between every pair of waypoints, and then solving the resulting ℓ -waypoint instance using the standard Bellman-Held-Karp dynamic programming algorithm [5, 31]. Klein and Marx [39] showed that if G is an undirected planar graph with polynomially bounded edge weights, then the problem can be solved significantly faster, in $2^{\mathcal{O}(\sqrt{\ell} \log \ell)} \cdot n^{\mathcal{O}(1)}$ time. It is also known that SUBSET TSP can be solved in $2^{\mathcal{O}(\sqrt{\ell} \log \ell)} \cdot n^{\mathcal{O}(1)}$ on edge-weighted directed planar graphs [46].

Amiri et al. [1] initiated the study of the variant of WRP discussed in this paper. They showed that the problem admits an XP algorithm w.r.t. the treewidth of the input graph. They also gave a randomized algorithm for the problem with running time $2^\ell \cdot n^{\mathcal{O}(1)}$, where ℓ is the number of waypoints; the algorithm can be derandomized if $\ell = \mathcal{O}((\log \log n)^{1/10})$. Schierreich and Suchý [52] improved the former algorithm by giving a deterministic algorithm with running time $2^{\mathcal{O}(\text{tw})} \cdot n$, which is tight under ETH [33]. Note that HAMILTONIAN PATH and therefore TSP is W[1]-hard w.r.t. the cliquewidth of the input graph [21].

It is somewhat surprising that, despite its popularity in theoretical computer science, the TSP problem did not receive much attention from the perspective of data reduction (kernelization). A notable exception is the work of Mannens et al. [43, 44] who studied MANY VISITS TSP and gave a polynomial kernel when the problem is parameterized by the vertex cover number of the input graph. In this work, we significantly expand the kernelization approach to TSP and its variants.

Structural parameters within the class of bounded treewidth. Even though all problems in FPT admit a kernel, not all of them admit a polynomial-sized kernel. It is not hard to see that TSP is AND-compositional and therefore we can (conditionally) exclude the existence of a polynomial kernel w.r.t. treewidth; more precisely, we show the following.

► **Lemma 1** (★). *There is no polynomial kernel for unweighted TRAVELING SALESPERSON PROBLEM with respect to either the fractioning number or the combined parameter treewidth and maximum degree, unless the polynomial hierarchy collapses.*

It follows that in order to obtain positive results we should investigate parameters within the class of bounded treewidth. Note that, roughly speaking, fractioning number is a *modulator to parameter-sized components*. A modulator is a set of vertices M such that when we remove M from a graph G the components of $G \setminus M$ fall into a specific (preferably simple) graph class (see Section 2 for formal definitions). Since almost all graph classes are closed under disjoint union, it is also popular to speak of addition of k vertices to a graph class (i.e., for a graph class \mathcal{G} we write $\mathcal{G} + kv$ if we are “adding” k vertices). Thus, it naturally models

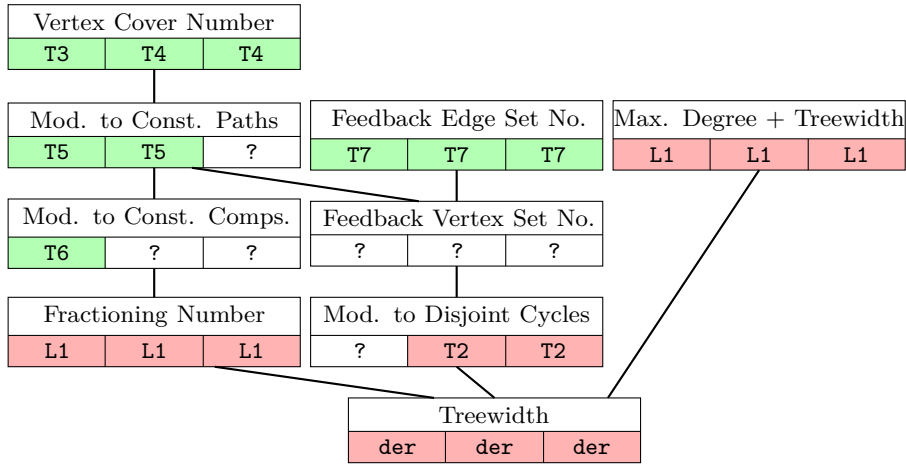


Figure 1 Overview of our results. Each node names a graph parameter and at its bottom it contains boxes representing (from left to right) TSP, SUBTSP, and WRP. If a node A is connected to a lower node B with an edge, then $B(G) = \mathcal{O}(A(G))$ for respective parameters. *Modulator*, *Constant*, and *Components* are shortened. Each box contains either a reference to the theorem which proves existence or non-existence of polynomial kernel (e.g. T2), or it contains ? if the problem is open, or it contains **der** if the result for such setting is derived from another result.

the situation when the input is close to a class of graphs where we can solve the problem efficiently (also called “Distance to \mathcal{G} ”). Modulator size is a popular structural parameter (see, e.g., [9, 41, 42]). Thus, the above result motivates us to study modulator-based parameters and their effect on kernelization of TSP and its variants. Indeed, many problems admit much more efficient algorithms w.r.t. vertex cover than for treewidth parameterization; see, e.g., [34, 19, 20]. Therefore, this is exactly the point where the positive part of our journey begins.

Before we do so, we discuss one more negative result. For SUBTSP we can, under a standard complexity-theoretic assumption, exclude the existence of a polynomial (Turing) kernel for the parameter size of a modulator to graphs of treewidth two. More specifically, we show the following.

► **Theorem 2** (★). *Unweighted SUBTSP with respect to the minimum k' such that there is a size k' modulator to cycles of size at most k' and there are at most k' non-terminals is WK[1]-hard.*

Therefore, SUBTSP parameterized by the size of a modulator to (disjoint) cycles does not admit a polynomial (Turing) kernel, unless all problems in WK[1] admit a polynomial Turing kernel. Moreover, it does not admit (classical) polynomial kernel, unless the polynomial hierarchy collapses. For more details on WK[1]-completeness, see [32].

Our Contribution. On a positive note, we begin with a polynomial kernel for TSP and WRP w.r.t. vertex cover number to demonstrate the general techniques we employ in this paper. We study the properties of a nice solution to the instance with respect to a particular structural parameter and its interaction with the modulator. We begin the study with TSP as a warm-up. In this case, we show that if there are “many” vertices in $G \setminus M$, where M is the modulator (vertex cover), then all but few of them behave in a “canonical way”. Based on the properties and cost of this canonical traversal, we identify which of these vertices can be safely discarded from the instance. In particular, we show the following.

► **Theorem 3** (★). *TRAVELING SALESPERSON PROBLEM admits a kernel with $\mathcal{O}(k^3)$ vertices, $\mathcal{O}(k^4)$ edges, and with a total bit-size $\mathcal{O}(k^{16})$, where k is the vertex cover number of G .*

In the case of SUBTSP or WRP, we cannot be sure that a solution visits all vertices in the modulator. Quite naturally, this results in the second kind of rule where we see that if there are “many” vertices attached to the modulator in the same way, we can mark some vertices in the modulator as terminals (there is a solution visiting them).

► **Theorem 4** (★). *WAYPOINT ROUTING PROBLEM admits a kernel with $\mathcal{O}(k^8)$ vertices, $\mathcal{O}(k^9)$ edges, and with a total bit-size $\mathcal{O}(k^{36})$, where k is the vertex cover number of G .*

Using similar, yet more involved results, we are able to find a polynomial kernel for SUBTSP for a modulator to constant-sized paths and for TSP for a modulator to constant-sized components.

One ingredient for our approach is that we may assume that all vertices outside the modulator are terminals, and therefore every solution visits them. It is not clear how to use our approach in the presence of capacities already in the case of modulator to paths, since the previous observation does not hold already in this simple case. Indeed, we show this by contracting some of the edges, which in the presence of capacities might have a side effect on the set of possible traversals of the component. The other and more important ingredient is the so-called blending of solutions (see Lemma 30).

► **Theorem 5** (★). *Let r be a fixed constant. SUBSET TSP admits a kernel of size $k^{\mathcal{O}(r)}$, where k is the size of a modulator to paths of size at most r .*

When we allow general constant-sized components outside the modulator, we have not yet succeeded in dealing with the highly connected non-terminals in the modulator. At this point, it is not clear whether many components with the same combination of canonical transversal and attachment to the modulator set ensure that we can safely mark a vertex in the modulator as a terminal.

► **Theorem 6** (★). *Let r be a fixed constant. TRAVELING SALESPERSON PROBLEM admits a kernel of size $k^{\mathcal{O}(r)}$, where k is the size of a modulator to components of size at most r .*

We conclude our algorithmic study by providing a rather simple polynomial kernel for WRP parametrized by the feedback edge set number. On the one hand, this result is an application of rather straightforward local reduction rules. On the other hand, these rules are heavily based on the use of edge capacities. It should be pointed out that we get a polynomial kernel for TSP as it has a polynomial compression to WRP, however, it can be shown that “local reduction rules” do not exist in the case of TSP.

► **Theorem 7** (★). *WAYPOINT ROUTING PROBLEM admits a kernel with $\mathcal{O}(k)$ vertices and edges and bit size $\mathcal{O}(k^4)$, where k is the feedback edge set number of G .*

For an overview of our results, please, refer to Figure 1.

Organization of the paper. We summarize the notation and technical results we rely on in Section 2. Section 3 contains a few useful technical lemmas and simple reduction rules. In Section 4, we begin with the core concepts applied in the case of TSP and vertex cover number. The similar approach is then applied in Section 5 to more general types of modulators; this is the most technical part of this manuscript. Finally, in Section 6 we conclude the results and discuss future research directions.

Statements where proofs or details are omitted due to space constraints are marked with ★. All missing proofs and details are available in the full version of the paper.

2 Preliminaries

We follow the basic notation of graph theory by Diestel [18]. In parameterized complexity theory, we follow the monograph of Cygan et al. [16].

A *walk* in a graph G is a non-empty alternating sequence of vertices and edges $S = v_1, e_1, \dots, e_{\ell-1}, v_\ell$ such that $v_i \in V(G)$, $e_i \in E(G)$, and $e_i = \{v_i, v_{i+1}\}$, $\forall i \in \{1, \dots, \ell-1\}$. It is *closed* if $v_\ell = v_1$. The weight of walk S is $\omega(S) = \sum_{i=1}^{\ell-1} \omega(e_i)$. If all vertices in a walk are distinct, it is called a *path*.

A *solution* to our problems is a closed walk S visiting every vertex in W of total weight at most b (traversing each edge e at most $\kappa(e)$ times). The least cost such walk is called an *optimal solution*. Given such a walk, we can construct the *corresponding multigraph* G_S which is a multigraph with vertex set being the set of vertices visited by S and each edge occurring as many times as traversed by S . We naturally extend ω to this multigraph, which yields $\omega(G_S) = \sum_{e \in E(G_S)} \omega(e) = \omega(S)$. The degree of a vertex in a multigraph is the number of the edges incident with it. Conversely, if a multigraph G_S is Eulerian (connected with all degrees even), then it admits a walk visiting every vertex of the graph and traversing each edge exactly as many times as occurring in G_S .

Structural Graph Parameters. Let $G = (V, E)$ be a graph. A set of edges $F \subseteq E$ is a *feedback edge set* of the graph G if $G \setminus F$ is an acyclic graph. *Feedback edge set number* $\text{fes}(G)$ of a graph G is the size of a smallest feedback edge set F of G . A set of vertices $C \subseteq V$ is called *vertex cover* of the graph G if it holds that $\forall e \in E$ we have $e \cap C \neq \emptyset$. The *vertex cover number* $\text{vc}(G)$ of a graph G is the least size of a vertex cover of G .

► **Definition 8.** Let \mathcal{G} be a graph family, let G be a graph, and $M \subseteq V(G)$. We say that M is a *modulator* of the graph G to the class \mathcal{G} if each connected component of $G \setminus M$ is in \mathcal{G} . The *distance* of G to \mathcal{G} , denoted $\text{modul}(G, \mathcal{G})$, is the minimum size of a modulator of G to \mathcal{G} .

Let r be a fixed constant. Let \mathcal{G}_{rc} be the class of graphs with every connected component having at most r vertices. The *distance of G to r -components* is $\text{modul}(G, \mathcal{G}_{rc})$. Similarly, if \mathcal{G}_{rp} is a class of graphs where every connected component is a path with at most r vertices, then the *distance of G to r -paths* is $\text{modul}(G, \mathcal{G}_{rp})$.

A set of vertices $C_r \subseteq V$ is called *r -fractioning set* of the graph G if $|C_r| \leq r$ and every connected component of $G \setminus C_r$ has at most r vertices. The *fractioning number* $\text{fn}(G)$ is a minimum $r \in \mathbb{N}$ such that there is an r -fractioning set C_r in graph G .

3 The Toolbox

In this section, we give formal proofs to a few technical statements we use throughout the rest of the paper. This yields a useful set of assumptions that allow us to present less technical proofs in the subsequent sections. We begin with a technical lemma we were not able to find in literature.

► **Lemma 9** (\star). *Let G be a graph with more than $2|V(G)| - 2$ edges. Then there is a cycle C in G such that the graph $G' = G \setminus E(C)$ has the same connected components as G .*

When proving the safeness of our reduction rules, it is easier to work with a solution that does not use many edges and traverses each edge at most twice. We show that we can always assume to work with such a solution.

► **Definition 10** (Nice Solution). *Let $(G, W, \omega, \kappa, \mathfrak{b})$ be an instance of TSP, SUBSET TSP, or WRP. We call a solution nice if it uses every edge at most twice and contains at most $2|V|$ edges (edge traversals) in total.*

Indeed, we may always assume that we work with a nice solution.

► **Lemma 11** (\star). *Let $(G, W, \omega, \kappa, \mathfrak{b})$ be an instance of TSP, SUBSET TSP, or WRP. There is a nice optimal solution.*

We continue with a remark about our instances. It is important to note that item (b) is applicable in general, however, one should be careful when doing so, since it increases the weights in the instance.

► **Remark 12** (\star). *Let $(G, W, \omega, \kappa, \mathfrak{b})$ be an instance of TSP, SUBSET TSP, or WRP. We assume that a) G is a connected graph and b) $\omega(e) > 0$ for all $e \in E(G)$.*

We apply the following lemma to reduce the weights of the kernelized instances. It follows in a rather straightforward way from results of Frank and Tardos [22].

► **Lemma 13** (\star). *There is a polynomial time algorithm, which, given an instance $(G, W, \omega, \kappa, \mathfrak{b})$ of TSP, SUBTSP, or WRP with at most d edges, produces ω' and \mathfrak{b}' such that the instances $(G, W, \omega, \kappa, \mathfrak{b})$ and $(G, W, \omega', \kappa, \mathfrak{b}')$ are equivalent and $(G, W, \omega', \kappa, \mathfrak{b}')$ is of total bit-size $\mathcal{O}(d^4)$.*

Finally, we present two simple reduction rules; we always assume that Reduction Rule 1 is not applicable.

► **Reduction Rule 1** (\star). *Let an instance of TSP, SUBTSP, or WRP be given. If $\mathfrak{b} < 0$, then answer no. Otherwise, if $|W| \leq 1$, then answer yes.*

► **Reduction Rule 2** (\star). *Let I be an instance of SUBSET TSP and $v \notin W$. For each pair of vertices $u, w \in N(v)$ we introduce a new edge $\{u, w\}$ into the graph with $\omega(\{u, w\}) = \omega(\{u, v\}) + \omega(\{v, w\})$ (if this creates parallel edges, then we only keep the one with the lower weight). Finally, we remove v together with all its incident edges.*

While Reduction Rule 2 is easy to apply, its application may destroy the structure of the input graph. Therefore, we only apply it to some specific vertices as explained in the subsequent sections.

4 Polynomial Kernel with Respect to Vertex Cover Number for TSP

In this (warm-up) section, we argue that TSP admits a polynomial kernel with respect to the vertex cover number. That is, we are going to present the most simple use-case of our reduction rules and therefore we can focus on the introduction of the core concept – the natural behavior. We begin with the definition of a (natural) behavior, which is a formal description of how a vertex “can behave” in a solution. Let M be a vertex cover of G of size $k = \text{vc}(G)$ and let $R = V \setminus M$.

► **Definition 14.** *For a vertex $r \in R$ a behavior of r is a multiset $F \subseteq \{\{r, m\} \mid \{r, m\} \in E, m \in M\}$ containing exactly two edges (edge occurrences). We let $B(r)$ be the set of all behaviors of r . We naturally extend the weight function such that for a behavior $F \in B(r)$ we set its weight to $\omega(F) = \sum_{e \in F} \omega(e)$. For a vertex $r \in R$ its natural behavior $b^{\text{nat}}(r)$ is a fixed minimizer of $\min \{\omega(F) \mid F \in B(r)\}$ that takes two copies of a minimum weight edge incident with r .*

The following lemma shows that in an optimal solution, most of the vertices of R actually use some behavior.

► **Lemma 15** (\star). *Let S be an optimal solution. Then the number of vertices $r \in R$ that are traversed more than once by S is at most k .*

Now, that we know what a behavior is, we can observe that each vertex can have one of two roles in the sought solution – they are either “just attached” using the natural behavior or they provide some connectivity between (two) vertices in the vertex cover. The role of a vertex is formalized as follows.

► **Definition 16**. *Let $r \in R$ and $F \in B(r)$ be a behavior of r . We say that F touches a vertex m of M if m has nonzero degree in $(\{r\} \cup M, F)$. We call the set $\text{imp}(F)$ of touched vertices the impact of behavior F .*

Let $\mathcal{I}_r = \{\text{imp}(F) \mid F \in B(r)\}$ be the set of all possible impacts of behaviors of r . Let $\mathcal{I} = \bigcup_{r \in R} \mathcal{I}_r$.

Note that we have $1 \leq |\text{imp}(F)| \leq 2$ for each behavior F . Therefore, we get $|\mathcal{I}| \leq \binom{k}{2} + k \leq k^2$. Note also that for each $r \in R$ we have $|\text{imp}(b^{\text{nat}}(r))| = 1$.

Now, we show that, in large instances, most of the vertices of R fall back to their natural behavior in an optimal solution. Towards this, we take a solution that differs from this in the fewest possible vertices $r \in R$. Then, we observe that if r is not in the natural behavior, then it is attached to (at least) two vertices in M . If there are many such “extra” edges in a solution, we can apply Lemma 9 – this would yield a contradiction. Thus, we get the following.

► **Lemma 17** (\star). *There is an optimal solution S such that for all but at most $3k$ vertices $r \in R$ the solution contains exactly the edges of $b^{\text{nat}}(r)$ among the edges incident with r .*

Now, we know that there are only a few vertices that behave unnaturally in an optimal solution, we would like to keep these in the kernel. We arrive at the question of which vertices to keep. To resolve this question, we observe that if a vertex deviates from its natural behavior, we (possibly) have to pay for this some extra price (which comes in the exchange of the provided connectivity). Thus, we would like to keep sufficiently many vertices for which this deviation is cheap. To this end, we first formalize the price.

► **Definition 18**. *For $r \in R$ and an impact $I \in \mathcal{I}$ let the price $P(r, I)$ of change from $b^{\text{nat}}(r)$ to I at r be $\omega(F_I) - \omega(b^{\text{nat}}(r))$ if there is a behavior $F_I \in B(r)$ with $\text{imp}(F_I) = I$ and we let it be ∞ otherwise.*

Note that if there is a behavior $F_I \in B(r)$ with $\text{imp}(F_I) = I$, then it is unique.

Now, based on this, we can provide the following reduction rule we employ.

► **Reduction Rule 3**. *For each $I \in \mathcal{I}$ if there are at most $3k$ vertices $r \in R$ with finite $P(r, I)$, then mark all of them. Otherwise mark $3k$ vertices $r \in R$ with the least $P(r, I)$.*

For each unmarked $r \in R$, remove r and decrease b by $\omega(b^{\text{nat}}(r))$.

Safeness. Let (G, ω, b) be the original instance and $(\widehat{G}, \widehat{\omega}, \widehat{b})$ be the new instance resulting from the application of the rule. Note that $\widehat{\omega}$ is just the restriction of ω to \widehat{G} which is a subgraph of G . Let R^- be the set of vertices of R removed by the rule. Note that $\widehat{b} = b - \sum_{r \in R^-} \omega(b^{\text{nat}}(r))$. We first show that if the new instance is a *yes*-instance, then so is the original one.

Let \widehat{S} be a solution walk in the new instance and let \widehat{G}_S be the corresponding multigraph formed by the edges of \widehat{S} . Note that the total weight of \widehat{G}_S is at most \widehat{b} . We construct a multigraph G_S by adding to \widehat{G}_S for each $r \in R^-$ the vertex r together with the edge set $b^{\text{nat}}(r)$. Since \widehat{G}_S is connected and each $b^{\text{nat}}(r)$ is incident with a vertex of $M \subseteq V(\widehat{G}_S)$, G_S is also connected. As each addition increases the degree of the involved vertex by exactly 2 and the degree of each vertex is even in \widehat{G}_S , it follows that the degree of each vertex is even in G_S . Hence G_S is Eulerian and contains all vertices of G . Since the weight of the added edges is exactly $\sum_{r \in R^-} \omega(b^{\text{nat}}(r))$ it follows that (G, ω, b) is a *yes*-instance.

Now suppose that the original instance (G, ω, b) is a *yes*-instance.

▷ **Claim 19** (\star). There is an optimal solution S for (G, ω, b) such that each $r \in R^-$ is incident exactly to the edges of $b^{\text{nat}}(r)$ in S .

Let S be a solution for (G, ω, b) as in the claim and G_S the corresponding multigraph. Let \widehat{G}_S be obtained from G_S by removing the edges of $b^{\text{nat}}(r)$ and vertex r for each $r \in R^-$. This reduces the total weight by exactly $\sum_{r \in R^-} \omega(b^{\text{nat}}(r))$, hence, as G_S is of weight at most b , \widehat{G}_S is of weight at most \widehat{b} . Furthermore, as each $r \in R^-$ has only one neighbor in G_S , \widehat{G}_S is connected. Moreover, each removal decreases the degree of exactly one remaining vertex by exactly 2, hence all degrees in \widehat{G}_S are even. Thus \widehat{G}_S is Eulerian, yielding a solution \widehat{S} for $(\widehat{G}, \widehat{\omega}, \widehat{b})$. This completes the proof. ◀

Since $|Z| \leq k^2$, the following observation is immediate.

▶ **Observation 20** (\star). After Reduction Rule 3 has been applied, the number of vertices in R is bounded by $3k^3$ and, hence, the total number of edges is $\mathcal{O}(k^4)$.

We conclude this section in the following theorem (which follows using Lemma 13).

▶ **Theorem 3** (\star). TRAVELING SALESPERSON PROBLEM admits a kernel with $\mathcal{O}(k^3)$ vertices, $\mathcal{O}(k^4)$ edges, and with a total bit-size $\mathcal{O}(k^{16})$, where k is the vertex cover number of G .

5 Polynomial Kernels with Respect to Modulator Size

In this section, we move to more general type of modulators. Recall that if M is a vertex cover, the connected components of $G \setminus M$ are single vertices. We are going to relax this a bit – we shall investigate general components of constant size and paths of constant size. For the first, more general one, we obtain a polynomial kernel for TSP. For the second, we obtain a polynomial kernel even for SUBTSP.

5.1 TSP and the Distance to Constant Size Components

For the rest of the section, we assume that we are given an undirected graph $G = (V, E)$, k is its distance to r -components, M the corresponding modulator, $E_M = \binom{M}{2}$, and $R = G \setminus M$. Let $M = \{m_1, \dots, m_k\}$ be a fixed order of the vertices of the modulator.

We begin with the definition of a behavior.

▶ **Definition 21.** For a connected component C of R we consider the graph $G_C = G[C \cup M] \setminus E_M$. A behavior of C is a multiset $F \subseteq E[G_C]$ of edges of G_C (each can be used at most twice) such that in the multigraph $(C \cup M, F)$, (i) each vertex $v \in C$ has nonzero even degree and (ii) each connected component contains a vertex of M , and the set F contains at most $2r$ edges incident with the vertices of M . We let $B(C)$ be the set of all behaviors of C . For a component C its natural behavior $b^{\text{nat}}(C)$ is a fixed minimizer of $\min \{\omega(F) \mid F \in B(C)\}$.

Now, it is not hard to see, that a solution might “visit” a component of $G \setminus M$ more than once. With this, we arrive at the key notion we introduce in this section – the segments. It is not hard to see that a solution may even revisit some terminals in a component; we shall later prove that this is the case for a few exceptional components only (see Lemma 25).

► **Definition 22.** Let C be a connected component of R and S be a walk starting and ending in $v_0 \in M$ forming a nice optimal solution to the instance. We split S into segments S_1, \dots, S_q such that each segment starts and ends in a vertex of M , whereas the inner vertices of each segment are from R . Let $\mathcal{F}(S, C)$ be obtained from the empty set by the following process: For each i , add S_i into $\mathcal{F}(S, C)$ if and only if there is a vertex $c \in C$ visited by S_i and not visited by any of the previous segments. Let $F(S, C)$ be the union of edges of S_i , $S_i \in \mathcal{F}(S, C)$, including multiplicities.

We observe that the multiset $F(S, C)$ is a behavior of C , i.e., $F(S, C) \in B(C)$.

Again, for a connected component of $G \setminus M$, we want to introduce the notion of touched vertices and the (connectivity) impact of a behavior of the component. For a better understanding of Definition 23, we refer the reader to an example in Figure 2.

► **Definition 23.** Let C be a connected component of R and $F \in B(C)$ be a behavior of C . We say that F touches a vertex m of M if m has nonzero degree in $(C \cup M, F)$. Let $T(F)$ be the set of touched vertices.

Consider the multiset $D(F)$ of edges obtained from the empty set as follows. For each connected component H of $(C \cup M, F)$ containing at least two vertices of M , let m_i be the vertex with the least index in $M \cap V(H)$. For each m_j in $(M \cap V(H)) \setminus \{m_i\}$ add to $D(F)$ a single edge $\{m_i, m_j\}$ if m_j is incident with an odd number of edges of F and a double edge $\{m_i, m_j\}$ if m_j is incident with an even number of edges of F .

We call the pair $\text{imp}(F) = (T(F), D(F))$ the impact of behavior F .

Let $\mathcal{I}_C = \{\text{imp}(F) \mid F \in B(C)\}$ be the set of all possible impacts of behaviors of C .

Let $\mathcal{I} = \bigcup_C \text{connected component of } R \mathcal{I}_C$.

From the Handshaking Lemma we get the following observation.

► **Observation 24.** Let m_i be as in Definition 23. Then m_i is incident with an even number of edges of $D(F)$ if and only if it is incident with an even number of edges of F .

It is not hard to see that $|\mathcal{I}| = k^{\mathcal{O}(r)}$.

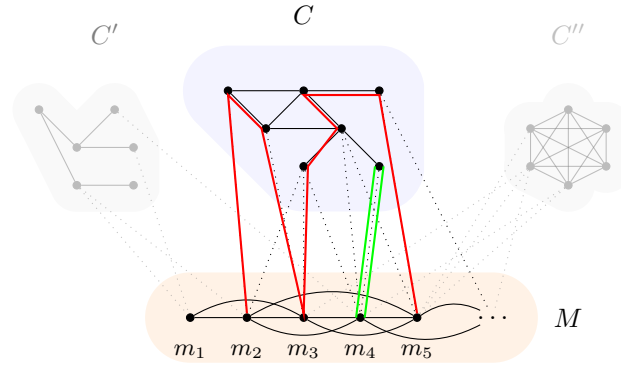
Now, we want to give some properties that an optimal solution should have. Namely, we want to see that most of the connected components of $G \setminus M$ fallback to their natural behavior. Towards this, we first prove that there are only a few segments that are not part of a behavior of any component; recall that if the solution “revisited” some terminals in the connected component, this segment might not be part of any behavior.

► **Lemma 25** (\star). There is an optimal solution S such that there are at most $2|M|$ segments of S that are not part of any $\mathcal{F}(S, C)$, and, furthermore, for all but at most $2|\mathcal{I}|^2 + 2|M|$ components we have $F(S, C) = b^{\text{nat}}(C)$ and $F(S, C)$ contains all edges of S incident with C .

Next, we want to introduce the notion of price of a transition between two impacts of a connected component.

► **Definition 26.** For a connected component C of R and an impact $I \in \mathcal{I}$ let $\mathcal{B}(C, I)$ be the set of behaviors $F \in B(C)$ satisfying $\text{imp}(F) = I$.

For a connected component C of R and a pair $I, I' \in \mathcal{I}$ let the price $P(C, I, I')$ of change from I to I' at C be $\min\{\omega(F) \mid F \in \mathcal{B}(C, I')\} - \omega(b^{\text{nat}}(C))$ if $\mathcal{B}(C, I')$ is non-empty and $I = \text{imp}(b^{\text{nat}}(C))$ and we let it be ∞ if some of the conditions is not met.



■ **Figure 2** Illustration of Definition 23. A behavior F consists of red and green edges, and vertices m_2, \dots, m_5 made the set $T(F)$ of touched vertices. We built the multiset $D(F)$ for behavior F as follows. We start with an empty set. For the red connected component, the vertex $m_i \in M$ with least index is m_2 . Thus, for m_3 we add to $D(F)$ double edge $\{m_2, m_3\}$, because the vertex m_3 is incident with even number of edges in behavior F . For the vertex m_5 , we add to $D(F)$ the edge $\{m_2, m_5\}$ only once, as the vertex m_5 is incident with odd number of edges in F . For the green connected component, we do not extend $D(F)$ since, in this connected component, there is only one vertex from M .

Now, we are ready to introduce the reduction rule used in this section. To keep a record of the vertices that are marked towards achieving a different goal, we mark them in different colors. *Red* vertices are those that behaves unnaturally in an optimal solution and their deviation from natural behavior is cheap. The task of the vertices marked in *green* is to ensure that at least one vertex with the same impact of natural behavior remains in the instance to provide the connectivity. Finally, there are components marked in *blue*. These are supposed to cover the segments which are not part of any behavior.

▶ **Reduction Rule 4** (\star). For each pair $I, I' \in \mathcal{I}$ if there are at most $2|\mathcal{I}|^2 + 2|M|$ components C with finite $P(C, I, I')$, then mark all of them in red. Otherwise mark $2|\mathcal{I}|^2 + 2|M|$ components C with the least $P(C, I, I')$ in red.

For each pair of vertices $u, v \in M$, if there is a component C in R such that G_C contains a u - v -path, then mark a component which contains the shortest such path in blue.

For each $I \in \mathcal{I}$, if there are unmarked components C with $\text{imp}(b^{\text{nat}}(C)) = I$, then do the following. If the number of such components is odd, then mark one arbitrary such component in green. If the number of such components is even, then mark two arbitrary such components in green.

For each unmarked component C , remove C from G and reduce \mathfrak{b} by $\omega(b^{\text{nat}}(C))$.

To prove Theorem 6 we estimate the number of vertices and edges in the reduced instance; Theorem 6 then follows using Lemma 13.

▶ **Lemma 27** (\star). After Reduction Rule 4 has been applied, the number of components is bounded by $2(|\mathcal{I}|^2 + 2|M|)|\mathcal{I}|^2 + \binom{|M|}{2} + 2|\mathcal{I}| = k^{\mathcal{O}(r)}$. Hence, the number of vertices and edges in the reduced graph G is $k^{\mathcal{O}(r)}$.

5.2 Subset TSP and the Distance to Constant Size Paths

We start by applying Reduction Rule 2 to all vertices of R . Note that after each application of the reduction rule R remains a disjoint union of paths, because each vertex has degree at most 2 within R . Hence, for the rest of the section we assume that $V(R) \subseteq W$.

22:12 On Polynomial Kernels for TSP

We reuse the notions of the (natural) behavior, impact, and price (Definitions 21, 23, and 26) from the previous section.

We define *piece* of $F \in B(C)$ as any connected component of $(C \cup M, F) \setminus M$ to which we add all incident edges in G_S . We define *legs* of a piece as a subset of its edges which are incident with vertices of M . As $G \setminus M$ consists of disjoint union of paths we note that each piece of F consists of a path on C and its legs.

► **Lemma 28** (\star). *Given a SUBSET TSP instance $(G, W, \omega, \mathfrak{b})$ let S be a nice optimal solution to the instance. Let k be the size of the modulator to disjoint union of paths. There are at most k components with pieces of $F(S, C)$ with more than 2 legs in behaviors $\{F(S, C) \mid \text{imp}(F(S, C)) = I\}$ for any fixed $I \in \mathcal{I}$.*

It is the case that every natural behavior has only two-legged pieces.

► **Observation 29** (\star). *For each C component of R , each piece of $b^{\text{nat}}(C)$ has two legs.*

The most technical part of this section is the following lemma that allows us to, under some conditions, mark a non-terminal in the modulator as a terminal. For this to work, we want to do the following. Take many components which share the impact of the natural behavior which touches the particular non-terminal. Since there are many, many of them also share the impact of the actual behavior and have all pieces 2-legged. For each of them we want to find a behavior that is half-way between the actual and the natural behavior (using the following lemma). This behavior should touch the non-terminal and be of at most the same weight as the actual behavior. Then we find two components for which the half-way behavior has the same impact and change these, so that we obtain a solution that visits the particular non-terminal.

► **Lemma 30** (Blending lemma \star). *Let $M' \subseteq M$ (the set of actually visited vertices) and C a component of R (therefore a path) such that $T(b^{\text{nat}}(C)) \not\subseteq M'$. Let $v \in T(b^{\text{nat}}(C)) \setminus M'$ and $A \in B(C)$ (the actual behavior) such that $T(A) \subseteq M'$ and such that each piece has two legs. Then there is a behavior $F \in B(C)$ such that $v \in T(F)$, $T(F) \subseteq T(A) \cup T(b^{\text{nat}}(C))$, every connected component of $(C \cup M, F)$ contains a vertex of M' , and $\omega(F) \leq \omega(A)$.*

Now, we are ready to present the reduction rule used in this case. The colors have the same meaning as in Reduction Rule 4. It remains to secure that every vertex in M that is incident with some natural behavior for many vertices in R is in W . Towards this, we prove that if a vertex $m \in M$ is needed for a natural behavior for many vertices in R , it is safe to mark m as a terminal. If this is not the case, we mark the vertices $r \in R$ in *yellow*.

► **Reduction Rule 5** (\star). *For each pair $I, I' \in \mathcal{I}$ of impacts if there are at most $2|\mathcal{I}|^2 + 2|M|$ components C with finite $P(C, I, I')$, then mark all of them in red. Otherwise mark $2|\mathcal{I}|^2 + 2|M|$ components C with the least $P(C, I, I')$ in red.*

For each pair of vertices $u, v \in M$, if there is a component C in R such that G_C contains a u - v -path, then mark the component which contains the shortest such path in blue.

For each $I \in \mathcal{I}$, if there are unmarked components C with $\text{imp}(b^{\text{nat}}(C)) = I$, then let $I = (T, D)$ and do the following. If $T \subseteq W$, then if the number of such components is odd, then mark one arbitrary such component in green. If the number of such components is even, then mark two arbitrary such components in green.

If $T \not\subseteq W$ and there are at most $((r+1)^{4r} \cdot 2^{4r+1} + k) \cdot |\mathcal{I}|$ unmarked components C with $\text{imp}(b^{\text{nat}}(C)) = I$, then mark them all in yellow.

If $T \not\subseteq W$ and there are more than $((r+1)^{4r} \cdot 2^{4r+1} + k) \cdot |\mathcal{I}|$ unmarked components C with $\text{imp}(b^{\text{nat}}(C)) = I$, then add T to W .

If W was not changed, then for each unmarked component C , remove C from G and reduce b by $\omega(b^{\text{nat}}(C))$.

To prove Theorem 5 we estimate the number of vertices and edges in the reduced instance; Theorem 5 then follows using Lemma 13.

► **Lemma 31** (★). *After Reduction Rule 5 has been applied, the number of components is bounded by $2(|\mathcal{I}|^2 + 2|M|)|\mathcal{I}|^2 + \binom{|M|}{2} + 2|\mathcal{I}| + ((r+1)^{4r} \cdot 2^{4r+1} + k) \cdot |\mathcal{I}|^2 = k^{\mathcal{O}(r)}$. Hence, the number of vertices and edges in the reduced graph G is $k^{\mathcal{O}(r)}$.*

6 Conclusions

The core focus of this work is kernelization of the TRAVELING SALESPERSON PROBLEM. To stimulate further research in this area we would like to promote some follow up research directions. Design of “local” rules might be impossible (as was mentioned in the case of feedback edge set number) and therefore we occasionally have to consider generalizations of this problem that give us more power when designing the reductions. An interesting open problem that remains in this area is whether TSP does admit a polynomial kernel with respect to the feedback vertex number. Towards this we have provided several steps that suggest it should be possible to design a polynomial kernel. It should be noted that in order to do so, one could try to “lift” our arguments to modulator to trees of any size. Note that this is not possible for cycles.

Yet another interesting line of work is the one of q -path vertex cover (q -pvc). This is a generalization of the vertex cover number – a graph G has q -pvc of size k if it has a modulator M (with $|M| = k$) such that in $G \setminus M$ there is no path of length q . It is not hard to see that some of our arguments can be applied for components of $G \setminus M$ that are stars (of arbitrary size). Therefore, it should be possible to give a polynomial kernel for 3-pvc. Thus, the question arises: Is there a polynomial kernel with respect to r -pvc for constant r ? In this case we are more skeptical and believe that the correct answer should be negative.

References

- 1 Saeed Akhoondian Amiri, Klaus-Tycho Foerster, and Stefan Schmid. Walking through waypoints. *Algorithmica*, 82(7):1784–1812, 2020. doi:10.1007/s00453-020-00672-z.
- 2 David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2011. doi:10.1515/9781400841103.
- 3 Sanjeev Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science, FOCS '96*, pages 2–11. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548458.
- 4 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, September 1998. doi:10.1145/290179.290180.
- 5 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.
- 6 Mark de Berg, Kevin Buchin, Bart M. P. Jansen, and Gerhard Woeginger. Fine-grained complexity analysis of two classic TSP variants. *ACM Transactions on Algorithms*, 17(1), December 2021. doi:10.1145/3414845.
- 7 Édouard Bonnet, Yoichi Iwata, Bart M. P. Jansen, and Łukasz Kowalik. Fine-grained complexity of k -OPT in bounded-degree graphs for solving TSP. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *Proceedings of the 27th Annual European Symposium on Algorithms, ESA '19*, volume 144 of *Leibniz International Proceedings in Informatics*, pages 23:1–23:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.23.

- 8 Sylvia C. Boyd and Robert Carr. Finding low cost TSP and 2-matching solutions using certain half-integer subtour vertices. *Discrete Optimization*, 8(4):525–539, 2011. doi:10.1016/j.disopt.2011.05.002.
- 9 Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003. doi:10.1016/S0166-218X(02)00242-1.
- 10 Robert D. Carr, Santosh S. Vempala, and Jacques Mandler. Towards a $4/3$ approximation for the asymmetric traveling salesman problem. In David B. Shmoys, editor, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '00*, pages 116–125. ACM/SIAM, 2000.
- 11 Timothy M. Chan and R. Ryan Williams. Deterministic APSP, Orthogonal Vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Transactions on Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926.
- 12 Miroslav Chlebík and Janka Chlebíková. Approximation hardness of travelling salesman via weighted amplifiers. In Ding-Zhu Du, Zhenhua Duan, and Cong Tian, editors, *Proceedings of the 25th International Conference on Computing and Combinatorics, COCOON '19*, volume 11653 of *Lecture Notes in Computer Science*, pages 115–127, Cham, 2019. Springer. doi:10.1007/978-3-030-26176-4_10.
- 13 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon University Pittsburgh, Management Sciences Research Group, 1976.
- 14 Gérard Cornuéjols, Jean Fonlupt, and Denis Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33(1):1–27, 1985. doi:10.1007/BF01582008.
- 15 G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, December 1958. doi:10.1287/opre.6.6.791.
- 16 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, Cham, 2015. doi:10.1007/978-3-319-21275-3.
- 17 Marek Cygan, Łukasz Kowalik, and Arkadiusz Socała. Improving TSP tours using dynamic programming over tree decompositions. *ACM Transactions on Algorithms*, 15(4), October 2019. doi:10.1145/3341730.
- 18 Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, Berlin, Heidelberg, 5th edition, 2017. doi:10.1007/978-3-662-53622-3.
- 19 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Information and Computation*, 209(2):143–153, 2011. doi:10.1016/j.ic.2010.11.026.
- 20 Jiří Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theoretical Computer Science*, 412(23):2513–2523, 2011. doi:10.1016/j.tcs.2010.10.043.
- 21 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010. doi:10.1137/080742270.
- 22 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/BF02579200.
- 23 David Gamarnik, Moshe Lewenstein, and Maxim Sviridenko. An improved upper bound for the TSP in cubic 3-edge-connected graphs. *Operations Research Letters*, 33(5):467–474, 2005. doi:10.1016/j.orl.2004.09.005.
- 24 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 25 Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In Rafail Ostrovsky, editor, *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science, FOCS '11*, pages 550–559. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.80.

- 26 Michel X. Goemans. Worst-case comparison of valid inequalities for the TSP. *Mathematical Programming*, 69:335–349, 1995. doi:10.1007/BF01585563.
- 27 Michelangelo Grigni, Elias Koutsoupias, and Christos H. Papadimitriou. An approximation scheme for planar graph TSP. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science, FOCS '95*, pages 640–645. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492665.
- 28 Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondřej Suchý. The parameterized complexity of local search for TSP, more refined. *Algorithmica*, 67(1):89–110, 2013. doi:10.1007/s00453-012-9685-8.
- 29 Gregory Gutin and Abraham P. Punnen, editors. *The traveling salesman problem and its variations*. Combinatorial Optimization. Springer, Boston, MA, 2007. doi:10.1007/b101971.
- 30 Arash Haddadan and Alantha Newman. Towards improving Christofides algorithm for half-integer TSP. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *Proceedings of the 27th Annual European Symposium on Algorithms, ESA '19*, volume 144 of *Leibniz International Proceedings in Informatics*, pages 56:1–56:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.56.
- 31 Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971. doi:10.1007/BF01584070.
- 32 Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (Turing) kernelization. *Algorithmica*, 71(3):702–730, 2015. doi:10.1007/s00453-014-9910-8.
- 33 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, March 2001. doi:10.1006/jcss.2000.1727.
- 34 Bart M. P. Jansen and Stefan Kratsch. Data reduction for graph coloring problems. *Information and Computation*, 231:70–88, 2013. doi:10.1016/j.ic.2013.08.005.
- 35 David S. Johnson and Lyle A. McGeoch. Experimental analysis of heuristics for the STSP. In Gregory Gutin and Abraham P. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, pages 369–443. Springer, Boston, MA, 2007. doi:10.1007/0-306-48213-4_9.
- 36 David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: a case study. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. Princeton University Press, 2018. doi:10.1515/9780691187563-011.
- 37 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. An improved approximation algorithm for TSP in the half integral case. 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 '20*, pages 28–39. ACM, 2020. doi:10.1145/3357713.3384273.
- 38 Philip N. Klein. A linear-time approximation scheme for planar weighted TSP. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, FOCS '05*, pages 647–657. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.7.
- 39 Philip N. Klein and Dániel Marx. A subexponential parameterized algorithm for subset TSP on planar graphs. In Chandra Chekuri, editor, *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '14*, pages 1812–1830. SIAM, 2014. doi:10.1137/1.9781611973402.131.
- 40 Giuseppe Lancia and Marcello Dalpasso. Algorithmic strategies for a fast exploration of the TSP 4-opt neighborhood. In Massimo Paolucci, Anna Sciomachen, and Pierpaolo Uberti, editors, *Advances in Optimization and Decision Science for Society, Services and Enterprises*, volume 3 of *AIRO Springer Series*, pages 457–470. Springer, Cham, 2019. doi:10.1007/978-3-030-34960-8_40.
- 41 Daniel Lokshantov, M. S. Ramanujan, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Brief announcement: Treewidth modulator: Emergency exit for DFVS. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming, ICALP '18*, volume 107 of *Leibniz International Proceedings in Informatics*, pages 110:1–110:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.110.

- 42 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Polynomial kernels for vertex cover parameterized by small degree modulators. *Theory of Computing Systems*, 62(8):1910–1951, 2018. doi:10.1007/s00224-018-9858-1.
- 43 Isja Mannens, Jesper Nederlof, Céline M. F. Swennenhuis, and Krisztina Szilágyi. On the parameterized complexity of the connected flow and many visits TSP problem. In Łukasz Kowalik, Michał Pilipczuk, and Paweł Rzazewski, editors, *Proceedings of the 47th International Workshop on Graph-Theoretic Concepts in Computer Science, WG '21*, volume 12911 of *Lecture Notes in Computer Science*, pages 52–79. Springer, 2021. doi:10.1007/978-3-030-86838-3_5.
- 44 Isja Mannens, Jesper Nederlof, Céline M. F. Swennenhuis, and Krisztina Szilágyi. On the parameterized complexity of the connected flow and many visits TSP problem. *CoRR*, abs/2106.11689, 2021. arXiv:2106.11689.
- 45 Dániel Marx. Searching the k -change neighborhood for TSP is $W[1]$ -hard. *Operations Research Letters*, 36(1):31–36, 2008. doi:10.1016/j.orl.2007.02.008.
- 46 Dániel Marx, Marcin Pilipczuk, and Michał Pilipczuk. On subexponential parameterized algorithms for Steiner tree and directed subset TSP on planar graphs. In Mikkel Thorup, editor, *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science, FOCS '18*, pages 474–484. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00052.
- 47 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, March 1999. doi:10.1137/S0097539796309764.
- 48 Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In Rafail Ostrovsky, editor, *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science, FOCS '11*, pages 560–569. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.56.
- 49 Marcin Mucha. $13/9$ -approximation for graphic TSP. In Christoph Dürr and Thomas Wilke, editors, *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science, STACS '12*, volume 14 of *Leibniz International Proceedings in Informatics*, pages 30–41. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.30.
- 50 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, February 1993. URL: <https://www.jstor.org/stable/3690150>.
- 51 Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, July 1976. doi:10.1145/321958.321975.
- 52 Šimon Schierreich and Ondřej Suchý. Waypoint routing on bounded treewidth graphs. *Information Processing Letters*, 173:106165, 2022. doi:10.1016/j.ipl.2021.106165.
- 53 András Sebő and Jens Vygen. Shorter tours by nicer ears: $7/5$ -approximation for the graph-TSP, $3/2$ for the path version, and $4/3$ for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014. doi:10.1007/s00493-014-2960-3.
- 54 David B. Shmoys and David P. Williamson. Analyzing the Held-Karp TSP bound: A monotonicity property with application. *Information Processing Letters*, 35(6):281–285, 1990. doi:10.1016/0020-0190(90)90028-V.

Maximizing Sums of Non-Monotone Submodular and Linear Functions: Understanding the Unconstrained Case

Kobi Bodek 

Department of Mathematics and Computer Science, Open University of Israel, Ra'anana, Israel

Moran Feldman  

Computer Science Department, University of Haifa, Israel

Abstract

Motivated by practical applications, recent works have considered maximization of sums of a submodular function g and a linear function ℓ . Almost all such works, to date, studied only the special case of this problem in which g is also guaranteed to be monotone. Therefore, in this paper we systematically study the simplest version of this problem in which g is allowed to be non-monotone, namely the unconstrained variant, which we term **Regularized Unconstrained Submodular Maximization (RegularizedUSM)**.

Our main algorithmic result is the first non-trivial guarantee for general **RegularizedUSM**. For the special case of **RegularizedUSM** in which the linear function ℓ is non-positive, we prove two inapproximability results, showing that the algorithmic result implied for this case by previous works is not far from optimal. Finally, we reanalyze the known Double Greedy algorithm to obtain improved guarantees for the special case of **RegularizedUSM** in which the linear function ℓ is non-negative; and we complement these guarantees by showing that it is not possible to obtain $(1/2, 1)$ -approximation for this case (despite intuitive arguments suggesting that this approximation guarantee is natural).

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis; Mathematics of computing \rightarrow Combinatorial optimization

Keywords and phrases Unconstrained submodular maximization, regularization, double greedy, non-oblivious local search, inapproximability

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.23

Related Version *Full Version*: <https://arxiv.org/pdf/2204.03412.pdf> [1]

Funding *Moran Feldman*: Research supported in part by Israel Science Foundation (ISF) grant number 459/20.

1 Introduction

The field of submodular optimization has been rapidly developing over the last two decades, partially due to new applications. Some of these applications have also motivated the optimization of composite objective functions that can be represented as the sum of a submodular function g and a linear function ℓ . Let us briefly discuss two such applications

The first application is *optimization with a regularizer*. To avoid overfitting in machine-learning, it is customary to optimize a function of the form $g - \ell$, where g is the quantity that we would like to maximize and ℓ is a (often linear) function that favors small solutions. This function ℓ is known as “regularizer” in the machine learning jargon, or “soft constraint” in the operations research jargon.

The other application we discuss is *optimization with a curvature*. Traditionally, the theoretical study of submodular optimization problems looks for approximation guarantees that apply to all submodular functions, or at least all monotone submodular functions. However, approximation guarantees of this kind are often pessimistic, and do not capture



© Kobi Bodek and Moran Feldman;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 23; pp. 23:1–23:17
Leibniz International Proceedings in Informatics



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

the practical performance of the algorithms analyzed. This has motivated studying how the optimal approximation ratios of various submodular maximization problems depend on various numerical function properties. Historically, the first property of this kind to be defined was the *curvature* property, which was suggested by Conforti and Cornuéjol [4] already in 1984. The curvature measures the distance of the submodular function from being linear, and a strong connection was demonstrated by Sviridenko et al. [17] between optimizing a submodular function with a given curvature and optimizing the sum $g + \ell$ of a monotone submodular function g and a linear function ℓ .

Motivated by the above applications, Sviridenko et al. [17] also initialized the study of the optimization of $g + \ell$ sums. In particular, they described algorithms with optimal approximation guarantees for this problem when g is a non-negative monotone submodular function, ℓ is a linear function and the optimization is subject to either a matroid or a cardinality constraint.¹ Later works obtained faster and semi-streaming algorithms for the same setting [7, 10, 11, 14]. However, in contrast to all these (often tight) results for monotone submodular functions g , much less is known about the case of non-monotone submodular functions. In fact, we are only aware of a single previous work that considered $g + \ell$ sums involving such functions [12].²

Given the rarity of results so far for optimizing $g + \ell$ with a function g that is non-monotone, this paper is devoted to a systematic study of the simplest problem of this kind, namely, unconstrained maximization of such sums. Formally, we study the **Regularized Unconstrained Submodular Maximization** (**RegularizedUSM**) problem. In this problem, we are given a non-negative submodular function $g: 2^{\mathcal{N}} \rightarrow \mathbb{R}_{\geq 0}$ and a linear function $\ell: 2^{\mathcal{N}} \rightarrow \mathbb{R}$ over the same ground set \mathcal{N} , and the objective is to output a set $T \subseteq \mathcal{N}$ maximizing the sum $g(T) + \ell(T)$. Unfortunately, it is not possible to prove standard multiplicative approximation ratios for **RegularizedUSM**.³ Therefore, we follow previous works (starting from [7, 17]), and look in this work for algorithms that output a (possibly randomized) set $T \subseteq \mathcal{N}$ such that $\mathbb{E}[g(T) + \ell(T)] \geq \max_{S \subseteq \mathcal{N}}[\alpha \cdot g(S) + \beta \cdot \ell(S)]$ for some coefficients $\alpha, \beta \geq 0$. For convenience, we say that an algorithm having this guarantee is an (α, β) -approximation algorithm.⁴

It is instructive to begin the study of **RegularizedUSM** with the special case in which the objective function g is guaranteed to be monotone (in addition to being non-negative and submodular). We refer below to this special case as “monotone **RegularizedUSM**”. The work of Feldman [7] on constrained maximization of $g + \ell$ immediately implies $(1 - e^{-\beta}, \beta)$ -approximation for monotone **RegularizedUSM** for every $\beta \in [0, 1]$. Our first result provides a matching inapproximability result.

► **Theorem 1.** *For every $\beta \geq 0$ and $\varepsilon > 0$, no polynomial time algorithm can guarantee $(1 - e^{-\beta} + \varepsilon, \beta)$ -approximation for monotone **RegularizedUSM** even when the linear function ℓ is guaranteed to be non-positive.*

¹ Technically, Sviridenko et al. [17] proved optimal approximation guarantees only for the case in which the coefficient β of ℓ is 1 (see details below). However, their results were extended to the general case of $\beta \geq 0$ by Feldman [7].

² Very recently, another work of this kind appeared as a pre-print [16]. However, the main result of [16] is identical to the result of [12]. In particular, it is important to note that the result of [16] applies only to non-positive ℓ functions, like the result of [12], although this is not explicitly stated in [16].

³ Formally, this is implied, e.g., by Theorem 1. Intuitively, the hurdle is that the combination of a positive g and a negative ℓ can lead to an optimal value that is very close to 0 compared to the values taken by the individual functions g and ℓ . When this happens, any algorithm with a positive multiplicative guarantee must output a solution that is close to optimal in terms of g and ℓ .

⁴ Some previous works compare their algorithms against $\alpha \cdot g(OPT) + \beta \cdot \ell(OPT)$, where OPT is a feasible set maximizing $g(OPT) + \ell(OPT)$, instead of comparing against $\max_{S \subseteq \mathcal{N}}[\alpha \cdot g(S) + \beta \cdot \ell(S)]$ like we do in this paper. This distinction is usually of little consequence.

We would like to draw attention to two properties of Theorem 1. First, for $\beta = 1$ the coefficient of g in the inapproximability proved by the theorem is $1 - 1/e$, matching the optimal approximation ratio for the problem of maximizing a monotone submodular function subject to a matroid constraint. Therefore, in a sense, adding the linear part ℓ makes the unconstrained problem as hard as this constrained problem. Interestingly, we get a similar result for `RegularizedUSM` below.

The other noteworthy property of Theorem 1 is that it applies to any $\beta \geq 0$, while the algorithmic result of Feldman [7] applies only to $\beta \in [0, 1]$. This difference between the results exists because, when ℓ can take positive values, setting the coefficient β to be larger than 1 might require the algorithm to output a set $T \subseteq \mathcal{N}$ obeying $\ell(T) > \max_{S \subseteq \mathcal{N}} \ell(S)$. However, it turns out that, when ℓ is non-positive, the algorithmic result can be extended to match Theorem 1 for every $\beta \geq 0$. To understand how this can be done, we need to discuss the previous work in a bit more detail.

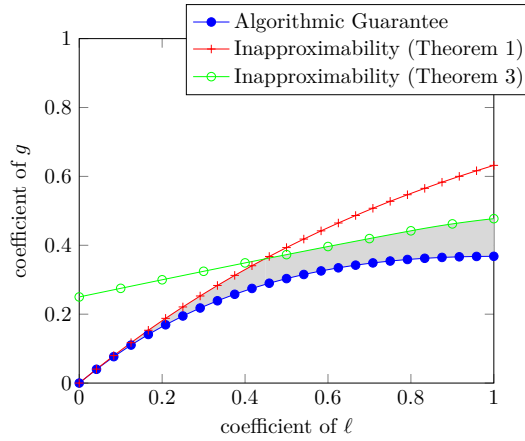
Sviridenko et al. [17] designed two algorithms for maximizing $g + \ell$ sums, one of which was based on the continuous greedy algorithm of Călinescu et al. [3]. It is possible to modify this algorithm to be based instead on a related algorithm called “measured continuous greedy” due to [8]. In general, this does not lead to any result for maximizing $g + \ell$ sums. However, Lu et al. [12] recently observed that one can obtain in this way results when ℓ is non-positive. In particular, it leads to $(1 - e^{-\beta}, \beta)$ -approximation for the special case of monotone `RegularizedUSM` in which ℓ is non-positive for any constant $\beta \geq 0$, which settles the approximability of monotone `RegularizedUSM`.

We now get to the study of (not necessarily monotone) `RegularizedUSM`. The only result that is known to date for this problem is $(1/e, 1)$ -approximation for the special case in which ℓ is non-positive, which was proved by Lu et al. [12] using the technique discussed above. Our main algorithmic contribution is the first algorithm with a non-trivial approximation guarantee for general `RegularizedUSM`.

► **Theorem 2.** *For every constant $\beta \in (0, 1]$, let us define $\alpha(\beta) = \beta(1 - \beta)/(1 + \beta)$. Then, for every constant $\varepsilon \in (0, \alpha(\beta))$, there exists a polynomial time $(\alpha(\beta) - \varepsilon, \beta - \varepsilon)$ -approximation algorithm for `RegularizedUSM`.*

We also study in more detail the special cases of `RegularizedUSM` in which ℓ is either non-negative or non-positive. The above mentioned result of Lu et al. [12] for `RegularizedUSM` with a non-positive ℓ can be extended (using the ideas of Feldman [7]) to get $(\beta e^{-\beta}, \beta)$ -approximation for the same special case for any $\beta \in [0, 1]$.⁵ It is not immediately clear, however, how good this extended result is. For example, one can compare it with the inapproximability result of Theorem 1 (which applies to the current setting as well), but there is a large gap between the above algorithmic and inapproximability results when the β coefficient of ℓ is relatively large (see Figure 1). This gap exists because Theorem 1 holds even in the special case in which g is monotone. Therefore, we prove the following theorem, which provides an alternative inapproximability result designed for the non-monotone case. Since it is difficult to understand the behavior of the expression stated in Theorem 3, we numerically draw it in Figure 1, which demonstrates that Theorem 3 closes much of the gap left with regard to `RegularizedUSM` with non-positive linear function ℓ .

⁵ Technically, this result can be extended to any constant $\beta \geq 0$, but this is not interesting since $\beta e^{-\beta}$ is a decreasing function for $\beta \geq 1$.



■ **Figure 1** Graphical presentation of the existing results for **RegularizedUSM** with a non-positive linear function ℓ . The x and y axes represent the coefficients of ℓ and g , respectively. The algorithmic guarantee drawn is the $(\beta e^{-\beta}, \beta)$ -approximation obtainable by generalizing Lu et al. [12]. The shaded area represents the gap that still exists between the best known approximation guarantee and inapproximability results.

▶ **Theorem 3.** Given a value $\beta \geq 0$, let us define

$$\alpha(\beta) = \min_{\substack{t \geq 1 \\ r \in (0, 1/2]}} \left\{ \frac{t+1 + \sqrt{(t+1)^2 - 8tr}}{4t} - \frac{r}{t+1} \left[1 - \beta - 2 \ln \left(\frac{t+1 - \sqrt{(t+1)^2 - 8tr}}{2} \right) \right] \right\}.$$

Then, for every $\varepsilon > 0$, no polynomial time algorithm guarantees $(\alpha(\beta) + \varepsilon, \beta)$ -approximation for **RegularizedUSM** even when the linear function ℓ is guaranteed to be non-positive.

It is interesting to note that, for $\beta = 1$, Theorem 3 matches the state-of-the-art inapproximability result of Oveis Gharan and Vondrák [15] for maximizing a non-negative submodular function subject to matroid constraint. Therefore, at least at the level of the known inapproximability results, **RegularizedUSM** with a non-positive ℓ is as hard as maximizing a non-negative submodular function subject to a matroid constraint.

It remains to consider the special case of **RegularizedUSM** with a non-negative ℓ . Here $g + \ell$ is a non-negative submodular function on its own right, and therefore, **RegularizedUSM** becomes a special case of the well-studied problem of Unconstrained Submodular Maximization (USM). The optimal approximation ratio for USM is $1/2$ due to an inapproximability result of Feige et al. [6], and the first algorithm to obtain this approximation ratio was the “Double Greedy” algorithm of Buchbinder et al. [2]. Specifically, Buchbinder et al. [2] described two variants of their algorithm, a deterministic variant guaranteeing $1/3$ -approximation, and a randomized variant guaranteeing $1/2$ -approximation. We refer below to these two variants as **DeterministicDG** and **RandomizedDG**, respectively. Interestingly, we are able to show in the next two theorems that the performance of **DeterministicDG** and **RandomizedDG** for **RegularizedUSM** is even better than what one would expect based on the guarantees of these algorithms for general USM.

▶ **Theorem 4.** When ℓ is non-negative, the algorithm **DeterministicDG** guarantees $(\alpha, 1 - \alpha)$ -approximation for **RegularizedUSM** for all $\alpha \in [0, 1/3]$ at the same time (the algorithm is oblivious to the value of α).

► **Theorem 5.** *When ℓ is non-negative, the algorithm *RandomizedDG* guarantees $(\alpha, 1 - \alpha/2)$ -approximation for *RegularizedUSM* for all $\alpha \in [0, 1/2]$ at the same time (the algorithm is oblivious to the value of α).*

We conclude this section with an interesting observation. Up to this point, the most well studied $g + \ell$ maximization problem was maximizing the sum of a non-negative monotone submodular function g and a linear function ℓ subject to a matroid constraint. When ℓ is positive, the optimal approximation guarantee for this problem is $(1 - 1/e, 1)$ [17], which is natural since $1 - 1/e$ is the optimal approximation ratio for maximizing such a function g subject to a matroid constraint [13]. Thus, one might expect to get $(1/2, 1)$ -approximation for *RegularizedUSM* with a non-negative ℓ . However, both Theorems 4 and 5 fail to prove such a guarantee, and we are able to show that this is not a coincidence.

► **Theorem 6.** *Even when the linear function ℓ is guaranteed to be non-negative, no polynomial time algorithm can guarantee $(1/2, 1)$ -approximation for *RegularizedUSM*.*

Paper Structure. In Section 2 we give a few formal definitions and explain the notation used throughout the paper. Then, we prove our inapproximability result for monotone *RegularizedUSM* (Theorem 1) in Section 3. Our results for general *RegularizedUSM*, *RegularizedUSM* with non-positive ℓ and *RegularizedUSM* with non-negative ℓ can be found in Sections 4, 5 and 6, respectively.

2 Preliminaries

Set Functions and Notation. Given a set function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}$, an element $u \in \mathcal{N}$ and a set $S \subseteq \mathcal{N}$, the marginal contribution of u to S with respect to f is $f(u | S) \triangleq f(S \cup \{u\}) - f(S)$. A set function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}$ is called *submodular* if it satisfies the intuitive property of diminishing returns. More formally, f is submodular if $f(u | S) \geq f(u | T)$ for every two sets $S \subseteq T \subseteq \mathcal{N}$ and element $u \in \mathcal{N} \setminus T$. An equivalent definition of submodularity is that f is submodular if $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$ for every two sets $S, T \subseteq \mathcal{N}$.

The set function f is called *monotone* if $f(S) \leq f(T)$ for every two sets $S \subseteq T \subseteq \mathcal{N}$, and it is called *linear* if there exist values $\{a_u \in \mathbb{R} \mid u \in \mathcal{N}\}$ such that $f(S) = \sum_{u \in S} a_u$ for every set $S \subseteq \mathcal{N}$.⁶ One can verify that any linear set function is submodular, but the reverse does not necessarily hold. Additionally, given a set S , a set function f and an element u , we often use $S + u$, $S - u$ and $f(u)$ as shorthands for $S \cup \{u\}$, $S \setminus \{u\}$ and $f(\{u\})$, respectively.

Multilinear extension. It is often useful to consider continuous extensions of set functions, and there are multiple ways in which this can be done. The proofs of our inapproximability results employ one such extension known as the *multilinear extension* (due to [3]). Formally, given a set function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}$, its multilinear extension is the function $F: [0, 1]^{\mathcal{N}} \rightarrow \mathbb{R}$ defined, for every vector $\mathbf{x} \in [0, 1]^{\mathcal{N}}$, by $F(\mathbf{x}) = \mathbb{E}[f(\mathbf{R}(\mathbf{x}))]$, where $\mathbf{R}(\mathbf{x})$ is a random subset of \mathcal{N} including every element $u \in \mathcal{N}$ with probability x_u , independently.

One can verify that, as is suggested by its name, the multilinear extension F is a multilinear function of the coordinates of its input vector. Furthermore, F is an extension of the set function f in the sense that for every set $S \subseteq \mathcal{N}$ we have $F(\mathbf{1}_S) = f(S)$, where $\mathbf{1}_S$ is the characteristic vector of the set S (i.e., a vector that has the value 1 in coordinates corresponding to elements of S , and the value 0 in the other coordinates).

⁶ Linear set functions are also known as *modular* functions.

Value Oracle. As is standard in the submodular optimization literature, we assume in this paper that algorithms access their set function inputs only through *value oracles*. A value oracle for a set function f is a black box that given a set $S \subseteq \mathcal{N}$ returns $f(S)$. One advantage of this convention is that it makes it possible to use information theoretic arguments to prove unconditional inapproximability results (i.e., inapproximability results that are not based on any complexity assumption). Nevertheless, if necessary, these inapproximability results can usually be adapted to apply also to succinctly represented functions (instead of functions accessed via value oracles) at the cost of introducing some complexity assumption [5].

3 Inapproximability for Monotone Functions

In this section we show an inapproximability for monotone **RegularizedUSM** (Theorem 1). All our inapproximability results in this paper are proved using Theorem 7. Since the proof of this theorem is a relatively straightforward adaptation of the symmetry gap framework of Vondrák [18], we defer it to the full version of this paper [1].

► **Theorem 7.** *Consider an instance (g, ℓ) of **RegularizedUSM** consisting of a non-negative submodular function $g: 2^{\mathcal{N}} \rightarrow \mathbb{R}_{\geq 0}$ and a linear function $\ell: 2^{\mathcal{N}} \rightarrow \mathbb{R}_{\geq 0}$, and assume that there exists a group \mathcal{G} of permutations over \mathcal{N} such that the equalities $g(S) = g(\sigma(S))$ and $\ell(S) = \ell(\sigma(S))$ hold for all sets $S \subseteq \mathcal{N}$ and permutations $\sigma \in \mathcal{G}$. Let G and L be the multilinear extensions of g and ℓ respectively, and for every vector $\mathbf{x} \in [0, 1]^{\mathcal{N}}$, let us denote $\bar{\mathbf{x}} = \mathbb{E}_{\sigma \in \mathcal{G}}[\mathbf{x}]$, i.e., $\bar{\mathbf{x}}$ is the expected vector $\sigma(\mathbf{x})$ when σ is picked uniformly at random out of \mathcal{G} . For any two constants $\alpha, \beta \geq 0$, if $\max_{S \subseteq \mathcal{N}}[\alpha \cdot g(S) + \beta \cdot \ell(S)]$ is strictly positive and*

$$\max_{\mathbf{x} \in [0, 1]^{\mathcal{N}}} [G(\bar{\mathbf{x}}) + L(\bar{\mathbf{x}})] \leq \max_{S \subseteq \mathcal{N}} [\alpha \cdot g(S) + \beta \cdot \ell(S)] ,$$

*then no polynomial time algorithm for **RegularizedUSM** can guarantee $((1 + \varepsilon)\alpha, (1 + \varepsilon)\beta)$ -approximation for any positive constant ε . Furthermore, this inapproximability guarantee holds also when we restrict attention to instances (g', ℓ') of **RegularizedUSM** having the following additional properties.*

- *If ℓ is non-negative or non-positive, then we can assume that ℓ' also has the same property.*
- *If g is monotone, then we can assume that g' is monotone as well.*

In the common case in which the linear function ℓ is a non-positive, the following observation allows us to produce slightly cleaner results using Theorem 7.

► **Observation 8.** *If ℓ is non-positive and $\alpha > 0$, then one can replace the term “ $((1 + \varepsilon)\alpha, (1 + \varepsilon)\beta)$ -approximation” in Theorem 7 with the term “ $(\alpha + \varepsilon, \beta)$ -approximation”.*

Proof. Theorem 7 proves, under some conditions, that no polynomial time algorithm for **RegularizedUSM** has $((1 + \varepsilon)\alpha, (1 + \varepsilon)\beta)$ -approximation. Furthermore, if we reduce the value of the constant parameter ε of the theorem by a factor of α , then the theorem also shows that no such algorithm can guarantee $(\alpha + \varepsilon, \beta + \varepsilon\beta/\alpha)$ -approximation. This implies the observation since, when ℓ is non-positive, any $(\alpha + \varepsilon, \beta)$ -approximation algorithm for **RegularizedUSM** is also an $(\alpha + \varepsilon, \beta + \varepsilon\beta/\alpha)$ -approximation algorithm. ◀

To prove Theorem 1 using Theorem 7, we need to define an instance \mathcal{I} of monotone **RegularizedUSM**. Specifically, consider a ground set \mathcal{N} of size $n \geq 2$ and a value $r \in (0, 1]$, and let us define

$$g(S) = \min\{|S|, 1\} \quad \text{and} \quad \ell(S) = -r \cdot |S| \quad \forall S \subseteq \mathcal{N} .$$

► **Lemma 9.** *For any constants $\varepsilon > 0$, $\beta \geq 0$ and $\alpha = 1 - e^{-\beta} + \varepsilon$, when n is large enough, there exists a value $r \in (0, 1]$ such that the inequality of Theorem 7 applies to \mathcal{I} and $\max_{S \subseteq \mathcal{N}}[\alpha \cdot g(S) + \beta \cdot \ell(S)]$ is strictly positive.*

Proof. Observe that $\max_{S \subseteq \mathcal{N}}[\alpha \cdot g(S) + \beta \cdot \ell(S)] \geq \alpha - \beta r = 1 - e^{-\beta} + \varepsilon - \beta r$ because S can be chosen as a singleton subset of \mathcal{N} . Let us now study the left hand side of the inequality of Theorem 7. Since both g and ℓ are unaffected when an arbitrary permutation is applied to the ground set, we can choose \mathcal{G} as the group of all permutations over \mathcal{N} . Thus, for every vector $\mathbf{x} \in [0, 1]^{\mathcal{N}}$, $\bar{\mathbf{x}} = \frac{\|\mathbf{x}\|_1}{n} \cdot \mathbf{1}_{\mathcal{N}}$. Therefore,

$$\begin{aligned} \max_{\mathbf{x} \in [0, 1]^{\mathcal{N}}} [G(\bar{\mathbf{x}}) + L(\bar{\mathbf{x}})] &= \max_{x \in [0, 1]} [G(x \cdot \mathbf{1}_{\mathcal{N}}) + L(x \cdot \mathbf{1}_{\mathcal{N}})] \\ &= \max_{x \in [0, 1]} [1 - (1 - x)^n - xrn] = 1 - r - r(n - 1)[1 - r^{1/(n-1)}] , \end{aligned}$$

where the last equality holds since the maximum is obtained for $x = 1 - \sqrt[n-1]{r}$. Note now that if we denote $y = (n - 1)^{-1}$, then by L'Hôpital's rule,

$$\lim_{n \rightarrow \infty} (n - 1)[1 - r^{1/(n-1)}] = \lim_{y \rightarrow 0} \frac{1 - r^y}{y} = \lim_{y \rightarrow 0} \frac{-r^y \ln r}{1} = -\ln r ,$$

and therefore, for a large enough n , $(n - 1)[1 - r^{1/(n-1)}] \geq -\ln r - \varepsilon$; which implies

$$\max_{\mathbf{x} \in [0, 1]^{\mathcal{N}}} [G(\bar{\mathbf{x}}) + L(\bar{\mathbf{x}})] \leq 1 - r - r[-\ln r - \varepsilon] \leq 1 - r(1 - \ln r) + \varepsilon.$$

Given the above bounds, we get that the inequality of Theorem 7 holds for any $r > 0$ obeying $1 - e^{-\beta} + \varepsilon - \beta r \geq 1 - r(1 - \ln r) + \varepsilon$. Since the last inequality is equivalent to $r - r \ln r \geq e^{-\beta} + \beta r$, it holds for $r = e^{-\beta} \subseteq (0, 1]$. Furthermore, for this choice of r ,

$$\max_{S \subseteq \mathcal{N}} [\alpha \cdot g(S) + \beta \cdot \ell(S)] \geq 1 - e^{-\beta} + \varepsilon - \beta r = 1 - (1 + \beta)e^{-\beta} + \varepsilon \geq 1 - \frac{1 + \beta}{1 + \beta} + \varepsilon = \varepsilon > 0 . \blacktriangleleft$$

Theorem 1, which we repeat here for convenience, now follows by combining Theorem 7, Observation 8 and Lemma 9 since g is a non-negative monotone submodular function and ℓ is a non-positive linear function.

► **Theorem 1.** *For every $\beta \geq 0$ and $\varepsilon > 0$, no polynomial time algorithm can guarantee $(1 - e^{-\beta} + \varepsilon, \beta)$ -approximation for monotone **RegularizedUSM** even when the linear function ℓ is guaranteed to be non-positive.*

4 Algorithm for the General Case

In this section we describe and analyze the only non-trivial algorithm known to date (as far as we know) for general **RegularizedUSM**. Using this algorithm we prove Theorem 2, which we repeat here for convenience.

► **Theorem 2.** *For every constant $\beta \in (0, 1]$, let us define $\alpha(\beta) = \beta(1 - \beta)/(1 + \beta)$. Then, for every constant $\varepsilon \in (0, \alpha(\beta))$, there exists a polynomial time $(\alpha(\beta) - \varepsilon, \beta - \varepsilon)$ -approximation algorithm for **RegularizedUSM**.*

Our algorithm is based on a non-oblivious local search, i.e., a local search guided by an auxiliary function rather than the objective function. Non-oblivious local searches have been used previously in the context of submodular maximization by, for example, Feige et al. [6] and Filmus and Ward [9]. The auxiliary function used by our algorithm is a function $h: 2^{\mathcal{N}} \rightarrow \mathbb{R}_{\geq 0}$ defined as follows. For every set $S \subseteq \mathcal{N}$,

$$h(S) = \mathbb{E}[g(S(\beta))] + \beta(1 + \beta) \cdot \ell(S) ,$$

where $S(\beta)$ is a random subset of S that includes every element of S with probability β , independently.

Ideally, we would like to find a local maximum with respect to h , i.e., a set $T \subseteq \mathcal{N}$ such that the value of $h(T)$ cannot be increased either by adding a single element to T , or by removing a single element from T . However, there are two issues that make the task of finding such a local maximum difficult.

- We do not know how to exactly evaluate the expectation in the definition of h in polynomial time. Therefore, whenever we need to calculate expressions involving h , we have to approximate them using sampling, which introduces estimation errors that have to be taken into account.
- A straightforward local search algorithm changes its current solution whenever adding or removing a single element improves this solution. However, the time complexity of such a naïve algorithm can be exponential. Therefore, our algorithm adds or removes an element only when this is beneficial enough, which means that the algorithm finds an approximate local maximum rather than a true one. Employing this idea is not trivial given the errors introduced by the sampling, as mentioned above. However, we manage to prove that, for the value Δ defined by our algorithm, with high probability: (i) the algorithm only makes changes that increase the value of $h(T)$ by $\Delta/2$ or more, and (ii) the algorithm continues to make changes as long as there exists some possible change that increases the value of $h(T)$ by at least $3\Delta/2$.

The quality of the approximate local maximum produced by our algorithm is controlled by the parameter ε of Theorem 2. Setting a lower value for ε decreases Δ , which increases our algorithm's time complexity, but also makes the approximate local maximum produced closer to being a true local maximum, and thus, improves the approximation guarantee.

Let \hat{S} be a subset of \mathcal{N} maximizing $(\alpha(\beta) - \varepsilon) \cdot g(\hat{S}) + (\beta - \varepsilon) \cdot \ell(\hat{S})$. To implement the solutions described in the last two bullets, it is useful to assume that the ground set \mathcal{N} does not include elements that have some problematic properties. The following reduction shows that we can assume that this is indeed the case without loss of generality. Due to space constraints, the proof of this reduction and some other proofs from this section are deferred to the full version of this paper [1]. In a nutshell, the first part of Reduction 10 is proved by arguing that elements violating this part cannot belong to the set we need to compete with, and thus, can be ignored; and the second part of the reduction is proved by showing that a simple algorithm outputting the best solution of size at most 1 achieves the guarantee of Theorem 2 when this part of the reduction is violated.

► **Reduction 10.** *While proving Theorem 2, we may assume that every element $u \in \mathcal{N}$ obeys*

$$\alpha(\beta) \cdot g(u) + \beta \cdot \ell(u) \geq 0 \quad \text{and} \quad \max\{g(u) + \ell(u), g(\emptyset)\} \leq \beta \cdot [g(\hat{S}) + \ell(\hat{S})] .$$

From this point until the end of the section, we denote by n the size of the ground set \mathcal{N} . We are now ready to describe our algorithm (given as Algorithm 1). This algorithm implicitly assumes that Reduction 10 was applied, that n is large enough and that $\max\{g(\emptyset), \max_{u \in \mathcal{N}} g(u)\} > 0$.⁷ The algorithm maintains a solution T , which it updates in

⁷ Let us explain why the problem becomes easy if either of the last two assumptions is violated. If n is bounded by a constant, it is possible to use exhaustive search to find the set $T \subseteq \mathcal{N}$ maximizing $g(T) + h(T)$, and one can verify that such a set has the properties guaranteed by Theorem 2. Additionally, if $\max\{g(\emptyset), \max_{u \in \mathcal{N}} g(u)\} = 0$, then the submodularity of g guarantees that g is the zero function, which means that we can get the guarantee of Theorem 2 by outputting the set $\{u \in \mathcal{N} \mid \ell(u) > 0\}$.

iterations. In each iteration, the algorithm calculates for every element u an estimate ω_u of the contribution of u to the g component of the auxiliary function h . Then, Line 6 of the algorithm looks for an element $u \in \mathcal{N} \setminus T$ which, based on the estimate ω_u , will increase $h(T)$ by Δ if added to T . If such an element u is found, the algorithm adds it to T and continues to the next iteration. Otherwise, Line 8 looks for an element $u \in T$ which will increase $h(T)$ by Δ if removed from T (again, based on the estimate ω_T). If such an element u is found, then the algorithm removes it from T and continues to the next iteration. However, if both Lines 6 and 8 fail to find an appropriate element, the algorithm assumes that it has encountered an approximate local maximum, and terminates. Somewhat surprisingly, when this happens the algorithm outputs a sample \hat{T} of $T(\beta)$ rather than the solution T itself (unless the value of this sample is negative, in which case the algorithm falls back to the solution \emptyset). We show below that if T is an approximate local maximum of the auxiliary function h , then $T(\beta)$ is in expectation a good solution with respect to the objective function.

■ **Algorithm 1** Non-oblivious Local Search (β, ε) .

```

1 Let  $\Delta \leftarrow \frac{\varepsilon}{2n} \cdot \max\{g(\emptyset), \max_{u \in \mathcal{N}} g(u)\}$  and  $T \leftarrow \{u \in \mathcal{N} \mid \ell(u) > 0\}$ .
2 for  $i = 1$  to  $\lceil 4n^2/\varepsilon \rceil + 1$  do
3   for every  $u \in \mathcal{N}$  do
4     Let  $\omega_u$  be an estimate of  $\beta \cdot \mathbb{E}[g(u \mid T(\beta) - u)]$  obtained by taking the average
       of  $\beta \cdot g(u \mid T(\beta) - u)$  for  $k = \lceil 128n^4\varepsilon^{-2}\beta^2 \cdot \ln(10n^4/\varepsilon) \rceil$  independent samples
       of  $T(\beta)$ .
5     if there exists  $u \in \mathcal{N} \setminus T$  such that  $\omega_u + \beta(1 + \beta) \cdot \ell(u) \geq \Delta$  then
6       Update  $T \leftarrow T + u$ .
7     else if there exists  $u \in T$  such that  $\omega_u + \beta(1 + \beta) \cdot \ell(u) \leq -\Delta$  then
8       Update  $T \leftarrow T - u$ .
9     else Exit the “for” loop.
10 Let  $\hat{T}$  be a sample of  $T(\beta)$ .
11 if  $g(\hat{T}) + \ell(\hat{T}) \geq 0$  then return  $\hat{T}$ .
12 else return  $\emptyset$ .
```

It is clear that Algorithm 1 runs in polynomial time, and therefore, we concentrate in the rest of this section on proving its approximation guarantee. Algorithm 1 makes multiple estimation during its execution. We say that an estimate ω_u is *good* if $|\omega_u - \mathbb{E}[g(u \mid T(\beta) - u)]| \leq \Delta/2$ (for the set T at the time in which the estimate was made), otherwise the estimate is *bad*. The following lemma is proved by showing that every estimate strongly concentrates due to the independent samples used to compute it, and then lower bounding the probability that all the estimates are good via the union bound.

► **Lemma 11.** *With high probability (a probability approaching 1 when n tends to infinity), all the estimates made by Algorithm 1 are good.*

Using the previous lemma, we can now prove that, with high probability, Algorithm 1 terminates with T being an approximate local maximum. Specifically, in the proof of the next lemma we show that if all the estimates are good (which is a high probability event by the previous lemma), then Algorithm 1 must encounter an approximate local maximum because otherwise the value of its solution grows to an impossibly high value.

► **Lemma 12.** *With high probability, when Algorithm 1 terminates we have*

$$h(T) \geq h(T + u) - 3\Delta/2 \quad \forall u \in \mathcal{N} \setminus T \quad \text{and} \quad h(T) \geq h(T - u) - 3\Delta/2 \quad \forall u \in T .$$

23:10 Maximizing Sums of Non-Monotone Submodular and Linear Functions

The last lemma shows that with high probability the final set T is an approximate local maximum with respect to h . Lemma 14 shows that this implies that $T(\beta)$ is a good solution in expectation. To prove Lemma 14, we need the following known lemma.

► **Lemma 13** (Lemma 2.2 of [6]). *Let $f: 2^X \rightarrow \mathbb{R}_{\geq 0}$ be a submodular function, and given a set $A \subseteq X$, let us denote by A_p a random subset of A where each element appears with probability $p \in [0, 1]$ (not necessarily independently). Then, $\mathbb{E}[f(A_p)] \geq (1-p) \cdot f(\emptyset) + p \cdot f(A)$.*

Recall that \hat{S} is a subset of \mathcal{N} maximizing the expression $(\alpha(\beta) - \varepsilon) \cdot g(\hat{S}) + (\beta - \varepsilon) \cdot \ell(\hat{S})$.

► **Lemma 14.** *If the set T obeys*

$$h(T) \geq h(T+u) - 3\Delta/2 \quad \forall u \in \mathcal{N} \setminus T \quad \text{and} \quad h(T) \geq h(T-u) - 3\Delta/2 \quad \forall u \in T ,$$

then $\mathbb{E}[g(T(\beta)) + \ell(T(\beta))] \geq (\alpha(\beta) - 3\varepsilon/4) \cdot g(\hat{S}) + (\beta - 3\varepsilon/4) \cdot \ell(\hat{S})$.

Proof. By the first part of Lemma 12, for every element $u \in \mathcal{N} \setminus T$, $h(T) \geq h(T+u) - 3\Delta/2$, or equivalently $h(u | T) \leq 3\Delta/2$. Therefore, by the submodularity of g ,

$$\begin{aligned} \mathbb{E}[g(T(\beta) \cup (\hat{S} \setminus T))] + (1+\beta) \cdot \ell(\hat{S} \setminus T) & \quad (1) \\ & \leq \mathbb{E}[g(T(\beta))] + \sum_{u \in \hat{S} \setminus T} \{\mathbb{E}[g(u | T(\beta) - u)] + (1+\beta) \cdot \ell(u)\} \\ & = \mathbb{E}[g(T(\beta))] + \beta^{-1} \cdot \sum_{u \in \hat{S} \setminus T} h(u | T) \leq \mathbb{E}[g(T(\beta))] + 3\beta^{-1} |\hat{S} \setminus T| \Delta/2 , \end{aligned}$$

where the equality holds because the law of total expectation implies that, for $u \notin T$,

$$\begin{aligned} h(u | T) & = \mathbb{E}[g((T+u)(\beta)) - g(T(\beta))] + \beta(1+\beta) \cdot \ell(u) \\ & = \beta \mathbb{E}[g(T(\beta) + u) - g(T(\beta))] + (1-\beta) \cdot \mathbb{E}[g(T(\beta)) - g(T(\beta))] + \beta(1+\beta) \cdot \ell(u) \\ & = \beta \mathbb{E}[g(T(\beta) + u) - g(T(\beta))] + \beta(1+\beta) \cdot \ell(u) = \beta \mathbb{E}[g(u | T(\beta) - u)] + \beta(1+\beta) \cdot \ell(u) . \end{aligned}$$

Similarly, since the second part of Lemma 12 implies that for every $u \in T$ we have $h(u | T - u) \geq -3\Delta/2$, the submodularity of g gives us

$$\begin{aligned} \mathbb{E}[g(T(\beta) \cap \hat{S})] - \beta(1+\beta) \cdot \ell(T \setminus \hat{S}) & \quad (2) \\ & \leq \mathbb{E}[g(T(\beta))] - \sum_{u \in T \setminus \hat{S}} \{\beta \cdot \mathbb{E}[g(u | T(\beta) - u)] - \beta(1+\beta) \cdot \ell(u)\} \\ & = \mathbb{E}[g(T(\beta))] - \sum_{u \in T \setminus \hat{S}} h(u | T - u) \leq \mathbb{E}[g(T(\beta))] + 3|T \setminus \hat{S}| \Delta/2 . \end{aligned}$$

Adding β times Inequality (1) to Inequality (2) now yields

$$\begin{aligned} \beta \cdot \mathbb{E}[g(T(\beta) \cup (\hat{S} \setminus T))] + \mathbb{E}[g(T(\beta) \cap \hat{S})] + \beta(1+\beta) \cdot [\ell(\hat{S} \setminus T) - \ell(T \setminus \hat{S})] & \quad (3) \\ \leq (1+\beta) \cdot \mathbb{E}[g(T(\beta))] + 3[|\hat{S} \setminus T| + |T \setminus \hat{S}|] \Delta/2 \leq (1+\beta) \cdot \mathbb{E}[g(T(\beta))] + 3n\Delta/2 . \end{aligned}$$

We can now use Lemma 13 to lower bound the first two terms on the leftmost side of the last inequality as follows.

$$\begin{aligned} \beta \cdot \mathbb{E}[g(T(\beta) \cup (\hat{S} \setminus T))] + \mathbb{E}[g(T(\beta) \cap \hat{S})] & \\ \geq \beta(1-\beta) \cdot g(\hat{S} \setminus T) + \beta^2 \cdot g(\hat{S} \cup T) + \beta \cdot g(T \cap \hat{S}) + (1-\beta) \cdot g(\emptyset) & \\ \geq \beta(1-\beta) \cdot [g(\hat{S} \setminus T) + g(T \cap \hat{S})] \geq \beta(1-\beta) \cdot g(\hat{S}) , & \end{aligned}$$

where the second inequality follows from the non-negativity of g , and the last inequality holds by g 's submodularity (and non-negativity). Plugging this inequality into Inequality (3) now gives

$$\beta(1 - \beta) \cdot g(\hat{S}) + \beta(1 + \beta) \cdot [\ell(\hat{S} \setminus T) - \ell(T \setminus \hat{S})] \leq (1 + \beta) \cdot \mathbb{E}[g(T(\beta))] + 3n\Delta/2 ,$$

and rearranging this inequality yields

$$\begin{aligned} \mathbb{E}[g(T(\beta)) + \ell(T(\beta))] &= \mathbb{E}[g(T(\beta))] + \beta \cdot \ell(T) \\ &\geq \frac{\beta(1 - \beta) \cdot g(\hat{S}) - 3n\Delta/2}{1 + \beta} + \beta \cdot [\ell(\hat{S} \setminus T) - \ell(T \setminus \hat{S})] + \beta \cdot \ell(T) \\ &\geq \alpha(\beta) \cdot g(\hat{S}) + \beta \cdot \ell(\hat{S}) - 3n\Delta/2 . \end{aligned}$$

To complete the proof of the lemma, it remains to show that $3n\Delta/2 \leq (3\varepsilon/4) \cdot [g(\hat{S}) + \ell(\hat{S})]$. Towards this goal, observe that

$$\max_{u \in \mathcal{N}} g(u) \leq \max_{u \in \mathcal{N}} \left\{ g(u) + \frac{1 + \beta}{2\beta^2} \cdot [\alpha(\beta) \cdot g(u) + \beta \cdot \ell(u)] \right\} = \frac{1 + \beta}{2\beta} \cdot \max_{u \in \mathcal{N}} [g(u) + \ell(u)] ,$$

where the inequality follows from the first part of Reduction 10. Using this inequality and the non-negativity of g , we can get

$$\begin{aligned} \frac{3n\Delta}{2} &= \frac{3\varepsilon}{4} \cdot \max\{g(\emptyset), \max_{u \in \mathcal{N}} g(u)\} \leq \frac{3\varepsilon(1 + \beta)}{8\beta} \cdot \max\{g(\emptyset), \max_{u \in \mathcal{N}} [g(u) + \ell(u)]\} \\ &\leq \frac{3\varepsilon(1 + \beta)}{8} \cdot [g(\hat{S}) + \ell(\hat{S})] \leq \frac{3\varepsilon}{4} \cdot [g(\hat{S}) + \ell(\hat{S})] , \end{aligned}$$

where the penultimate inequality follows from the second part of Reduction 10, and the last inequality uses the observation that the second part of Reduction 10 and the non-negativity of g imply together that $g(\hat{S}) + \ell(\hat{S})$ is non-negative. \blacktriangleleft

We are now ready to prove Theorem 2.

Proof of Theorem 2. Recall that \hat{T} is a sample of $T(\beta)$ for the value of the set T when Algorithm 1 terminates. Lemmata 12 and 14 prove together that there exists a high probability event \mathcal{E} such that

$$\mathbb{E}[g(\hat{T}) + \ell(\hat{T}) \mid \mathcal{E}] \geq (\alpha(\beta) - 3\varepsilon/4) \cdot g(\hat{S}) + (\beta - 3\varepsilon/4) \cdot \ell(\hat{S}) .$$

The last two lines of Algorithm 1 guarantee that this algorithm always outputs a set whose value is at least $g(\hat{T}) + h(\hat{T})$ because $g(\emptyset) + \ell(\emptyset) = g(\emptyset) \geq 0$. Therefore, if we denote by \bar{T} the set outputted by Algorithm 1, then we also have

$$\mathbb{E}[g(\bar{T}) + \ell(\bar{T}) \mid \mathcal{E}] \geq (\alpha(\beta) - 3\varepsilon/4) \cdot g(\hat{S}) + (\beta - 3\varepsilon/4) \cdot \ell(\hat{S}) .$$

The last two lines of Algorithm 1 also guarantee that the output set \bar{T} of Algorithm 1 always has a non-negative value, and therefore, $\mathbb{E}[g(\bar{T}) + h(\bar{T}) \mid \bar{\mathcal{E}}] \geq 0$. Combining this inequality with the previous one using the law of total expectation yields

$$\begin{aligned} \mathbb{E}[g(\bar{T}) + h(\bar{T})] &\geq \Pr[\mathcal{E}] \cdot \mathbb{E}[g(\bar{T}) + \ell(\bar{T}) \mid \mathcal{E}] \\ &\geq (1 - o(1)) \cdot [(\alpha(\beta) - 3\varepsilon/4) \cdot g(\hat{S}) + (\beta - 3\varepsilon/4) \cdot \ell(\hat{S})] \\ &\geq (\alpha(\beta) - \varepsilon) \cdot g(\hat{S}) + (\beta - \varepsilon) \cdot \ell(\hat{S}) = \max_{S \subseteq \mathcal{N}} [(\alpha(\beta) - \varepsilon) \cdot g(S) + (\beta - \varepsilon) \cdot \ell(S)] , \end{aligned}$$

23:12 Maximizing Sums of Non-Monotone Submodular and Linear Functions

where the second inequality holds since $g(\bar{T}) + h(\bar{T})$ is always non-negative, the equality follows from the definition of \hat{S} , and $o(1)$ represents a term that diminishes when n goes to infinity. To justify the third inequality, note that

$$\begin{aligned} o(1) \cdot [(\alpha(\beta) - 3\varepsilon/4) \cdot g(\hat{S}) + (\beta - 3\varepsilon/4) \cdot \ell(\hat{S})] &\leq o(1) \cdot (\beta - 3\varepsilon/4) \cdot [g(\hat{S}) + \ell(\hat{S})] \\ &\leq (\varepsilon/4) \cdot [g(\hat{S}) + \ell(\hat{S})] , \end{aligned}$$

where the last inequality here holds for large enough values of n because the second part of Reduction 10 and the non-negativity of g imply together that $g(\hat{S}) + \ell(\hat{S})$ is non-negative. ◀

5 Inapproximability for Negative Linear Functions

In this section we prove Theorem 3, which we repeat here for convenience.

► **Theorem 3.** *Given a value $\beta \geq 0$, let us define*

$$\alpha(\beta) = \min_{\substack{t \geq 1 \\ r \in (0, 1/2]}} \left\{ \frac{t+1 + \sqrt{(t+1)^2 - 8tr}}{4t} - \frac{r}{t+1} \left[1 - \beta - 2 \ln \left(\frac{t+1 - \sqrt{(t+1)^2 - 8tr}}{2} \right) \right] \right\} .$$

*Then, for every $\varepsilon > 0$, no polynomial time algorithm guarantees $(\alpha(\beta) + \varepsilon, \beta)$ -approximation for **RegularizedUSM** even when the linear function ℓ is guaranteed to be non-positive.*

The proof of Theorem 3 is based on Theorem 7, and therefore, we start this proof by describing an instance \mathcal{I} of **RegularizedUSM**. This instance is very similar to the instance used by Oveis Gharan and Vondrák [15] to prove their hardness result for maximizing a non-negative (not necessarily monotone) submodular function subject to a matroid constraint. Specifically, the instance \mathcal{I} has 3 parameters: an integer $n \geq 1$, a real value $t \geq 1$ and a real value $r \in (0, 1/2]$. The ground set of \mathcal{I} is $\mathcal{N} = \{a, b\} \cup \{a_i, b_i \mid i \in [n]\}$, and its objective functions are $\ell(S) = -r \cdot |S \cap \{a_i, b_i \mid i \in [n]\}|$ and

$$\begin{aligned} g(S) = t \cdot (|S \cap \{a, b\}| \bmod 2) &+ \mathbf{1}[a \notin S] \cdot \mathbf{1}[S \cap \{a_i \mid i \in [n]\} \neq \emptyset] \\ &+ \mathbf{1}[b \notin S] \cdot \mathbf{1}[S \cap \{b_i \mid i \in [n]\} \neq \emptyset] . \end{aligned}$$

One can verify that g is indeed a non-negative submodular function. Additionally, the functions g and ℓ are both symmetric in the sense that the following types of swaps do not affect the values of these functions.

- Any swap of the identities of the elements of $\{a_i \mid i \in [n]\}$.
- Swapping the identifies of a with b plus swapping the identities of a_i and b_i for every $i \in [n]$.

Let \mathcal{G} be the group of permutations obtained by combining swaps of these two kinds in any way.

In the next lemma, G and L are the multilinear extensions of g and ℓ , respectively, and $\bar{\mathbf{x}} = \mathbb{E}_{\sigma \in \mathcal{G}}[\sigma(\mathbf{x})]$. Theorem 3 follows by combining this lemma with Theorem 7 and Observation 8. The (quite technical) proof of Lemma 15 can be found in the full version of this paper [1].

► **Lemma 15.** *Let r and t be the values for which the maximum is obtained in the definition of $\alpha(\beta)$. Then, for any constant $\varepsilon > 0$ and a large enough n , $\max_{S \subseteq \mathcal{N}}[(\alpha(\beta) + \varepsilon) \cdot g(S) + \beta \cdot \ell(S)]$ is strictly positive and $\max_{\mathbf{x} \in [0,1]^{\mathcal{N}}} [G(\bar{\mathbf{x}}) + L(\bar{\mathbf{x}})] \leq \max_{S \subseteq \mathcal{N}}[\alpha(\beta) \cdot g(S) + \beta \cdot \ell(S)]$.*

6 Results for Positive Linear Functions

In this section we study `RegularizedUSM` in the special case in which the linear function ℓ is non-negative. As explained in Section 1, following related known results, it is natural to expect a $(1/2, 1)$ -approximation for this case since $1/2$ is the best possible approximation ratio for unconstrained maximization of a non-negative submodular function. However, we show in Section 6.1 that this cannot be done (Theorem 6).

Let us now define $f \triangleq g + \ell$. As explained in Section 1, since f is a non-negative submodular function on its own right, one can optimize it using any algorithm for `Unconstrained Submodular Maximization (USM)`. The first algorithm to obtain a tight approximation ratio of $1/2$ for USM was an algorithm called “Double Greedy” due to Buchbinder et al. [2]. Buchbinder et al. [2] described two variants of their algorithm, a deterministic variant that we term `DeterministicDG` and guarantees $1/3$ -approximation, and a randomized variant that we term `RandomizedDG` and guarantees $1/2$ -approximation. It should also be noted that the original analysis of [2] proves slightly stronger results than the above stated approximation ratios. Specifically, their analysis shows that `DeterministicDG` always outputs a set of value at least $\frac{1}{3}[f(S) + f(\emptyset) + f(\mathcal{N})] \geq \frac{1}{3}g(S) + \frac{2}{3}\ell(S)$ for any set S , where the inequality holds because ℓ is non-negative; which implies that `DeterministicDG` is a $(1/3, 2/3)$ -approximation algorithm. Similarly, the analysis of Buchbinder et al. [2] shows that `RandomizedDG` outputs a set whose expected value is at least $\frac{1}{4}[2f(S) + f(\emptyset) + f(\mathcal{N})] \geq \frac{1}{2}g(S) + \frac{3}{4}\ell(S)$, which implies that `RandomizedDG` is a $(1/2, 3/4)$ -approximation algorithm.

Theorems 4 and 5 show that `DeterministicDG` and `RandomizedDG`, respectively, guarantee (α, β) -approximation for many additional pairs of α and β . The proofs of these theorems can be found in Sections 6.2 and 6.3, respectively.

6.1 Impossibility of the Naturally Expected Approximation Guarantee

In this section we prove the following theorem. We note that the technique used in the proof of this theorem can also prove a somewhat stronger result. However, since the improvement represented by this stronger result is not very significant, we chose to state in the theorem the cleaner and more conceptually important result rather than the strongest result achievable.

► **Theorem 6.** *Even when the linear function ℓ is guaranteed to be non-negative, no polynomial time algorithm can guarantee $(1/2, 1)$ -approximation for `RegularizedUSM`.*

The proof of Theorem 6 is based on Theorem 7, and therefore, we need to describe an instance \mathcal{I} of `RegularizedUSM` that has an integer parameter $n \geq 2$. The ground set of the instance \mathcal{I} is $\mathcal{N} = \{a, b\} \cup \{c_i \mid i \in [n]\}$, and its objective functions are given, for every $S \subseteq \mathcal{N}$, by $\ell(S) = 1/3$ and

$$g(S) = 2 \cdot [(S \cap \{a, b\}) \bmod 2] + \mathbf{1}[\{a, b\} \cap S \neq \emptyset] \cdot \mathbf{1}[\{c_i \mid i \in [n]\} \not\subseteq S] .$$

One can verify that g is indeed a non-negative submodular function. Additionally, the functions g and ℓ are both symmetric in the sense that swapping the identities of a and b does not change the values of these functions for any set, and the same applies to any swap of the identities of the elements of $\{c_i \mid i \in [n]\}$. Let \mathcal{G} be the group of permutations obtaining by combining swaps of these two kinds in any way.

In the next lemma, G and L are the multilinear extensions of g and ℓ , respectively, and $\bar{\mathbf{x}} = \mathbb{E}_{\sigma \in \mathcal{G}}[\sigma(\mathbf{x})]$. By combining this lemma with Theorem 7, we get that, even when the linear function ℓ is non-negative, no polynomial time algorithm for `RegularizedUSM` can guarantee $(0.4998(1 + \varepsilon) + (n - 1.0003)(1 + \varepsilon)/(n - 1))$ -approximation for any $\varepsilon > 0$ and large enough n . Theorem 6 follows by choosing $\varepsilon = 0.0003/n$. Due to space constraints, (the quite technical) proof of Lemma 16 has been deferred to the full version of this paper [1].

► **Lemma 16.** For a large enough n ,

$$\max_{\bar{\mathbf{x}} \in [0,1]^{\mathcal{N}}} [G(\bar{\mathbf{x}}) + L(\bar{\mathbf{x}})] \leq \max_{S \subseteq \mathcal{N}} \left[0.4998 \cdot g(S) + \frac{n - 1.0003}{n - 1} \cdot \ell(S) \right],$$

and the right hand side of the inequality is strictly positive.

6.2 Reanalysis of Deterministic Double Greedy

In this section we prove Theorem 4, which we repeat here for convenience. The algorithm `DeterministicDG` referred to by this theorem is given as Algorithm 2 (recall that $f \triangleq g + \ell$).

► **Theorem 4.** When ℓ is non-negative, the algorithm *DeterministicDG* guarantees $(\alpha, 1-\alpha)$ -approximation for *RegularizedUSM* for all $\alpha \in [0, 1/3]$ at the same time (the algorithm is oblivious to the value of α).

Algorithm 2 DeterministicDG.

```

1 Denote the elements of  $\mathcal{N}$  by  $u_1, u_2, \dots, u_n$  in an arbitrary order.
2 Let  $X_0 \leftarrow \emptyset$  and  $Y_0 \leftarrow \emptyset$ .
3 for  $i = 1$  to  $n$  do
4   Let  $a_i \leftarrow f(u_i \mid X_{i-1})$  and  $b_i \leftarrow -f(u_i \mid Y_{i-1} - u_i)$ .
5   if  $a_i \geq b_i$  then Let  $X_i \leftarrow X_{i-1} + u_i$  and  $Y_i \leftarrow Y_{i-1}$ .
6   else Let  $X_i \leftarrow X_{i-1}$  and  $Y_i \leftarrow Y_{i-1} - u_i$ .
7   return  $X_n (= Y_n)$ .
```

The heart of the proof of Theorem 4 is the following lemma. To state this lemma, we need to define, for every integer $0 \leq i \leq n$ and set $S \subseteq \mathcal{N}$, $S^{(i)} = (S \cup X_i) \cap Y_i$.

► **Lemma 17.** For every integer $1 \leq i \leq n$, value $\alpha \in [0, 1/3]$ and set $S \subseteq \mathcal{N}$, $\alpha \cdot [f(X_i) - f(X_{i-1})] + (1 - 2\alpha) \cdot [f(Y_i) - f(Y_{i-1})] \geq \alpha \cdot [f(S^{(i-1)}) - f(S^{(i)})]$.

Before we get to the proof of Lemma 17, let us show why it implies Theorem 4.

Proof of Theorem 4. Fix some $\alpha \in [0, 1/3]$ and set $S \subseteq \mathcal{N}$. Summing up Lemma 17 over all integer $1 \leq i \leq n$, we get

$$\alpha \cdot \sum_{i=1}^n [f(X_i) - f(X_{i-1})] + (1 - 2\alpha) \cdot \sum_{i=1}^n [f(Y_i) - f(Y_{i-1})] \geq \alpha \cdot \sum_{i=1}^n [f(S^{(i-1)}) - f(S^{(i)})].$$

The sums in the last inequality are telescopic sums, and collapsing them yields

$$\alpha \cdot [f(X_n) - f(X_0)] + (1 - 2\alpha) \cdot [f(Y_n) - f(Y_0)] \geq \alpha \cdot [f(S^{(0)}) - f(S^{(n)})].$$

One can observe that $X_n = Y_n = S^{(n)}$, $f(X_0) = g(\emptyset) \geq 0$, $f(Y_0) = g(\mathcal{N}) + \ell(\mathcal{N}) \geq \ell(S)$ and $S^{(0)} = S$. Plugging all these observations into the previous inequality yields

$$\alpha \cdot f(X_n) + (1 - 2\alpha) \cdot [f(X_n) - \ell(S)] \geq \alpha \cdot [f(S) - f(X_n)].$$

It remains to rearrange the last inequality, and plug in $f(S) = g(S) + \ell(S)$, which implies $f(X_n) \geq \alpha \cdot g(S) + (1 - \alpha) \cdot \ell(S)$. The theorem now follows since: (i) X_n is the output set of Algorithm 2, and (ii) the last inequality holds for every $\alpha \in [0, 1/3]$ and set $S \subseteq \mathcal{N}$. ◀

Let us now prove Lemma 17.

Proof of Lemma 17. Buchbinder et al. [2] showed that Algorithm 2 guarantees⁸

$$[f(X_i) - f(X_{i-1})] + [f(Y_i) - f(Y_{i-1})] \geq f(S^{(i-1)}) - f(S^{(i)}) . \quad (4)$$

Furthermore, we prove below that we also have the inequality

$$f(Y_i) - f(Y_{i-1}) \geq 0 . \quad (5)$$

These two inequalities imply the lemma together since the inequality guaranteed by the lemma is equal to $\alpha \cdot (4) + (1 - 3\alpha) \cdot (5)$ – note that the coefficients α and $1 - 3\alpha$ in this expression are non-negative for the range of possible values for α .

It remains to prove Inequality (5). If $Y_i = Y_{i-1}$, then Inequality (5) trivially holds as an equality. Consider now the case of $Y_i \neq Y_{i-1}$. By Lines 5 and 6 of Algorithm 2, this case happens only when $b_i > a_i$, and Y_i is set to $Y_{i-1} - u_i$ when this happens. Therefore, we get in this case $f(Y_i) - f(Y_{i-1}) = f(Y_{i-1} - u_i) - f(Y_{i-1}) = b_i > \frac{a_i + b_i}{2} \geq 0$, where the last inequality holds since Buchbinder et al. [2] also showed that $a_i + b_i \geq 0$. ◀

6.3 Reanalysis of Randomized Double Greedy

In this section we prove Theorem 5, which we repeat here for convenience. The algorithm `RandomizedDG` referred to by this theorem is given as Algorithm 3 (recall that $f \triangleq g + \ell$).

► **Theorem 5.** *When ℓ is non-negative, the algorithm `RandomizedDG` guarantees $(\alpha, 1 - \alpha/2)$ -approximation for `RegularizedUSM` for all $\alpha \in [0, 1/2]$ at the same time (the algorithm is oblivious to the value of α).*

Algorithm 3 `RandomizedDG`.

```

1 Denote the elements of  $\mathcal{N}$  by  $u_1, u_2, \dots, u_n$  in an arbitrary order.
2 Let  $X_0 \leftarrow \emptyset$  and  $Y_0 \leftarrow \emptyset$ .
3 for  $i = 1$  to  $n$  do
4   Let  $a_i \leftarrow f(u_i \mid X_{i-1})$  and  $b_i \leftarrow -f(u_i \mid Y_{i-1} - u_i)$ .
5   if  $b_i \leq 0$  then Let  $X_i \leftarrow X_{i-1} + u_i$  and  $Y_i \leftarrow Y_{i-1}$ .
6   else if  $a_i \leq 0$  then Let  $X_i \leftarrow X_{i-1}$  and  $Y_i \leftarrow Y_{i-1} - u_i$ .
7   else
8     with probability  $\frac{a_i}{a_i + b_i}$  do Let  $X_i \leftarrow X_{i-1} + u_i$  and  $Y_i \leftarrow Y_{i-1}$ .
9     otherwise Let  $X_i \leftarrow X_{i-1}$  and  $Y_i \leftarrow Y_{i-1} - u_i$ . // Occurs with prob.  $\frac{b_i}{a_i + b_i}$ 
10  return  $X_n (= Y_n)$ .
```

The heart of the proof of Theorem 5 is the next lemma. To state this lemma, we need to define, like in Section 6.2, $S^{(i)} = (S \cup X_i) \cap Y_i$ for every integer $0 \leq i \leq n$ and set $S \subseteq \mathcal{N}$.

► **Lemma 18.** *For every integer $1 \leq i \leq n$, value $\alpha \in [0, 1/2]$ and set $S \subseteq \mathcal{N}$, $(\alpha/2) \cdot \mathbb{E}[f(X_i) - f(X_{i-1})] + (1 - 3\alpha/2) \cdot \mathbb{E}[f(Y_i) - f(Y_{i-1})] \geq \alpha \cdot \mathbb{E}[f(S^{(i-1)}) - f(S^{(i)})]$.*

⁸ Technically, Buchbinder et al. [2] proved Inequality (4) only for the special case in which S is a set maximizing f . However, their analysis does not use this property.

23:16 Maximizing Sums of Non-Monotone Submodular and Linear Functions

Before we get to the proof of Lemma 18, let us show why it implies Theorem 5.

Proof of Theorem 5. Fix some $\alpha \in [0, 1/2]$ and set $S \subseteq \mathcal{N}$. Summing up Lemma 18 over all integer $1 \leq i \leq n$, we get

$$\frac{\alpha}{2} \sum_{i=1}^n \mathbb{E}[f(X_i) - f(X_{i-1})] + (1 - 3\alpha/2) \cdot \sum_{i=1}^n \mathbb{E}[f(Y_i) - f(Y_{i-1})] \geq \alpha \cdot \sum_{i=1}^n \mathbb{E}[f(S^{(i-1)}) - f(S^{(i)})] .$$

Due to the linearity of the expectation, the sums in the last inequality are telescopic sums. Collapsing these sums yields

$$\frac{\alpha}{2} \mathbb{E}[f(X_n) - f(X_0)] + (1 - 3\alpha/2) \cdot \mathbb{E}[f(Y_n) - f(Y_0)] \geq \alpha \cdot \mathbb{E}[f(S^{(0)}) - f(S^{(n)})] .$$

Observe now that, like in the proof of Theorem 4, we have $X_n = Y_n = S^{(n)}$, $f(X_0) = g(\emptyset) \geq 0$, $f(Y_0) = g(\mathcal{N}) + \ell(\mathcal{N}) \geq \ell(S)$ and $S^{(0)} = S$. Plugging all these observations into the previous inequality yields

$$\frac{\alpha}{2} \mathbb{E}[f(X_n)] + (1 - 3\alpha/2) \cdot \mathbb{E}[f(X_n) - \ell(S)] \geq \alpha \cdot \mathbb{E}[f(S) - f(X_n)] .$$

It remains to rearrange the last inequality, and plug in $f(S) = g(S) + \ell(S)$, which implies $\mathbb{E}[f(X_n)] \geq \alpha \cdot g(S) + (1 - \alpha/2) \cdot \ell(S)$. The theorem now follows since: (i) X_n is the output set of Algorithm 2, and (ii) the last inequality holds for every $\alpha \in [0, 1/2]$ and set $S \subseteq \mathcal{N}$. ◀

Let us now prove Lemma 18.

Proof of Lemma 18. Buchbinder et al. [2] showed that Algorithm 3 guarantees⁹

$$\mathbb{E}[f(X_i) - f(X_{i-1})] + \mathbb{E}[f(Y_i) - f(Y_{i-1})] \geq 2\mathbb{E}[f(S^{(i-1)}) - f(S^{(i)})] . \quad (6)$$

Given this inequality, to prove the lemma it suffices to show that $(1 - 2\alpha) \cdot \mathbb{E}[f(Y_i) - f(Y_{i-1})] \geq 0$ (because adding this inequality to $\alpha/2$ times Inequality (6) yields the inequality that we want to prove). Below we prove the stronger claim that the inequality $f(Y_i) \geq f(Y_{i-1})$ holds deterministically. One observe that this stronger claim indeed implies $(1 - 2\alpha) \cdot \mathbb{E}[f(Y_i) - f(Y_{i-1})]$ because $1 - 2\alpha$ is non-negative in the range of allowed values for α .

If $Y_i = Y_{i-1}$, then the inequality $f(Y_i) \geq f(Y_{i-1})$ trivially holds as an equality. Therefore, we assume from now on $Y_i \neq Y_{i-1}$, which implies $Y_i = Y_{i-1} - u_i$. Due to the condition in Line 5 of Algorithm 3, Y_i can be set to $Y_{i-1} - u_i$ only when $b_i > 0$, and thus, $f(Y_i) = f(Y_{i-1} - u_i) = f(Y_{i-1}) + b_i > f(Y_{i-1})$. ◀

References

- 1 Kobi Bodek and Moran Feldman. Maximizing sums of non-monotone submodular and linear functions: Understanding the unconstrained case. *CoRR*, abs/2204.03412, 2022. doi:10.48550/arXiv.2204.03412.
- 2 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM J. Comput.*, 44(5):1384–1402, 2015. doi:10.1137/130929205.
- 3 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.

⁹ Again, the proof of [2] was technically stated only for the case in which S is a set maximizing f , but it extends without modification to any set $S \subseteq \mathcal{N}$.

- 4 Michele Conforti and Gérard Cornuéjols. Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the rado-edmonds theorem. *Discret. Appl. Math.*, 7(3):251–274, 1984. doi:10.1016/0166-218X(84)90003-9.
- 5 Shahar Dobzinski and Jan Vondrák. From query complexity to computational complexity. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC)*, pages 1107–1116. ACM, 2012. doi:10.1145/2213977.2214076.
- 6 Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM J. Comput.*, 40(4):1133–1153, 2011. doi:10.1137/090779346.
- 7 Moran Feldman. Guess free maximization of submodular and linear sums. *Algorithmica*, 83(3):853–878, 2021. doi:10.1007/s00453-020-00757-9.
- 8 Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 570–579. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.46.
- 9 Yuval Filmus and Justin Ward. Monotone submodular maximization over a matroid via non-oblivious local search. *SIAM J. Comput.*, 43(2):514–542, 2014. doi:10.1137/130920277.
- 10 Chris Harshaw, Moran Feldman, Justin Ward, and Amin Karbasi. Submodular maximization beyond non-negativity: Guarantees, fast algorithms, and applications. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 2634–2643. PMLR, 2019. URL: <http://proceedings.mlr.press/v97/harshaw19a.html>.
- 11 Ehsan Kazemi, Shervin Minaee, Moran Feldman, and Amin Karbasi. Regularized submodular maximization at scale. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 5356–5366. PMLR, 2021. URL: <http://proceedings.mlr.press/v139/kazemi21a.html>.
- 12 Cheng Lu, Wenguo Yang, and Suixiang Gao. Regularized non-monotone submodular maximization. *CoRR*, abs/2103.10008, 2021. arXiv:2103.10008.
- 13 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.
- 14 Sofia Maria Nikolakaki, Alina Ene, and Evimaria Terzi. An efficient framework for balancing submodularity and cost. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao, editors, *The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1256–1266. ACM, 2021. doi:10.1145/3447548.3467367.
- 15 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In Dana Randall, editor, *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1098–1116. SIAM, 2011. doi:10.1137/1.9781611973082.83.
- 16 Xin Sun, Dachuan Xu, Yang Zhou, and Chenchen Wu. Maximizing modular plus non-monotone submodular functions. *CoRR*, abs/2203.07711, 2022. arXiv:2203.07711.
- 17 Maxim Sviridenko, Jan Vondrák, and Justin Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. *Math. Oper. Res.*, 42(4):1197–1218, 2017. doi:10.1287/moor.2016.0842.
- 18 Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM J. Comput.*, 42(1):265–304, 2013. doi:10.1137/110832318.

List Colouring Trees in Logarithmic Space

Hans L. Bodlaender  

Utrecht University, The Netherlands

Carla Groenland  

Utrecht University, The Netherlands

Hugo Jacob  

ENS Paris-Saclay, France

Abstract

We show that LIST COLOURING can be solved on n -vertex trees by a deterministic Turing machine using $O(\log n)$ bits on the worktape. Given an n -vertex graph $G = (V, E)$ and a list $L(v) \subseteq \{1, \dots, n\}$ of available colours for each $v \in V$, a list colouring for G is a proper colouring c such that $c(v) \in L(v)$ for all v .

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Mathematics of computing \rightarrow Trees; Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases List colouring, trees, space complexity, logspace, graph algorithms, tree-partition-width

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.24

Related Version *Full Version:* <https://arxiv.org/abs/2206.09750>

Funding *Carla Groenland:* Supported by the project CRACKNP that has received funding from the European Research Council (grant agreement No 853234).

Acknowledgements We thank Marcin Pilipczuk and Michał Pilipczuk for discussions.

1 Introduction

Various applications can be modelled as an instance of LIST COLOURING, e.g., the vertices may correspond to communication units, with lists giving the possible frequencies or channels that a vertex may choose from as colours and edges showing which units would interfere if they are assigned the same colour [17, 24].

Given a graph $G = (V, E)$ and given a list $L(v)$ of colours for each vertex $v \in V$, an L -colouring c is a proper colouring (that is, $c(u) \neq c(v)$ when $uv \in E$) mapping every vertex v to a colour in the list $L(v)$. This gives rise to the following computational problem.

LIST COLOURING

Input: A graph $G = (V, E)$ with a list $L(v) \subseteq \{1, \dots, n\}$ of available colours for each $v \in V$.

Question: Is there an L -colouring for G ?

LIST COLOURING is computationally hard. It is NP-complete on cographs [19] and on planar bipartite graphs, even when all lists are of size at most 3 [18]. The problem remains hard when parameterised by “tree-like” width measures: it was first shown to be $W[1]$ -hard parameterised by treewidth in 2011 by [16] and recently shown to be XNLP-hard implying $W[t]$ -hardness for all t by [4]. On the other hand, on n -vertex trees the problem can be solved in time linear in n (using hashing) [19], but this algorithm may use $\Omega(n)$ space.



© Hans L. Bodlaender, Carla Groenland, and Hugo Jacob;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 24; pp. 24:1–24:15

Leibniz International Proceedings in Informatics



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

In this paper, we study the auxiliary space requirements of LIST COLOURING on trees in terms of the number of vertices n of the tree. We assume that the vertices of T have been numbered $1, \dots, n$, which gives a natural order on them, and that, given vertices v, v' in T and $i, i' \in \{1, \dots, n\}$, it can be checked in $O(\log n)$ space whether the i th colour in $L(v)$ equals the i' th colour in $L(v')$. As is usual for the complexity class L (logspace), we measure the space requirements in terms of the number of bits on the work tape of a deterministic Turing machine, where the description of the tree and the lists are written on a (read-only) input tape. In particular, the number of bits on the input tape is allowed to be much larger.

Since n -vertex trees have pathwidth $O(\log n)$, our problem can be solved non-deterministically using $O(\log^2 n)$ bits on the work tape (see Proposition 3). However, doing better than this is surprisingly challenging, even in the non-deterministic case! Our main result is as follows.

► **Theorem 1.** LIST COLOURING for trees is in L.

Our initial interest in the space complexity of LIST COLOURING on trees arose from a recent result showing that LIST COLOURING parameterised by pathwidth is XNLP-complete [4]. XNLP is the class of problems on an input of size n with parameter k , which can be solved by a non-deterministic Turing machine in $f(k)n^{O(1)}$ time and $f(k)\log n$ space for some computable f . Since the treewidth of a graph is upper bounded by the pathwidth, LIST COLOURING is also XNLP-hard parameterised by treewidth. This is conjectured¹ to imply that there is a constant k^* for which any deterministic Turing machine needs $\omega(\log n)$ space in order to solve LIST COLOURING for n -vertex graphs of treewidth k^* ; this work shows that $k^* > 1$. It seems likely that LIST COLOURING parameterised by treewidth is not in XNLP, and we conjecture that it is complete for a parameterised analogue of NAuxPDA (also known as SAC) from [1, 23]. Considering such classes which (also) have space requirements (complexity classes such as XL, XNL and XNLP [4, 5, 7, 15]) has proven successful in classifying the complexity of parameterised problems which are not known to be complete for any classes that only consider time requirements. Since some of such classes are very naturally modelled by instances of LIST COLOURING, we believe the complexity class of LIST COLOURING on trees could be of theoretical interest as well.

Another motivation for studying space requirements comes from practice, since memory can be much more of a bottleneck than processing time (e.g. for dynamic programming approaches). This motivates the development of techniques to reduce the space complexity. Although many techniques have been established to provide algorithms which are efficient with respect to time, fewer techniques are known to improve the space complexity. Notable exceptions include the logspace analogue of Bodlaender's and Courcelle's theorem [14] which allows one to check any monadic second-order formula on graphs of bounded treewidth in logspace (which in particular allows one to test membership in any minor-closed family) and Reingold's [25] work on undirected connectivity. Reachability and isomorphism questions have also been well-studied on restricted graph classes, e.g. [8, 21, 22]. Another interesting piece of related work [13] shows that for each graph H , LIST H -COLOURING is either in L or NL-hard. We remark that LIST H -COLOURING on trees (for fixed H) is easily seen to be solvable in logspace using an analogue of Proposition 3 or the logspace Courcelle's theorem [14]. The difficulty in our case comes from the fact that the sizes of the lists are unbounded.

¹ The conjecture (see [23, Conjecture 2.1] or [4, Conjecture 5.1]) states that there is no deterministic algorithm for an XNLP-hard problem that runs in XP time and FPT space. If for each k , there exists a constant $f(k)$ such that LIST COLOURING can be solved in space $f(k)\log n$ on n -vertex graphs of treewidth k , then this would in particular yield an algorithm running in $n^{f(k)}$ time and $f(k)n^{O(1)}$ space.

We also generalise our algorithm to graphs of bounded tree-partition-width (also called strong treewidth).

► **Corollary 2.** *There is a deterministic $O(k \log k \log n)$ space algorithm for LIST COLOURING on n -vertex graphs with a given tree-partition of width k .*

The algorithm of Corollary 2 does not run in FPT time. We also include a simple proof that LIST COLOURING is $W[1]$ -hard when parameterised by tree-partition-width, which shows that it is unlikely that there exists an algorithm running in FPT time.

The algorithm of Theorem 1 is highly non-trivial and requires several conceptual ideas that we have attempted to separate out by first explaining some key ideas and an easier deterministic algorithm that uses $O(\log^2 n)$ space in Section 3. We assume our given tree to be rooted and transverse it by first “recursing” on children whose subtrees are not the largest. This bounds the number of such recursions by $O(\log n)$, and so we can “spend” $O(\log n)$ space per recursion. When we move to the heaviest subtree, we have either already rejected, or may forget entirely about the colour of v , or found a single colour that “works” for the non-heavy subtrees of v (“criticality”). Along a heavy branch, we always keep at most two such colours “per recursion depth” (the colour of v , and possibly one of its parent; once we move to the child of v we may forget the colour of the parent of v).

There are two further main ideas that remove the additional $\log n$ -factor. Most importantly, when we “move” from a vertex v to one of its children u , we will let the amount of storage allocated for storing v and its colour depend on the size of the subtree of u : the larger the subtree, the less space we allow. In the extreme case in which the size of the subtree of u is linear in the size of the subtree of v , we allow only a *constant* number of bits. At specific points during the algorithm, when more space is available temporarily, we use this stored information to recompute the vertex v and its colour. Suppose that v has d children u_1, \dots, u_d . We define the list $L_j(v)$ as the colours c such that when we give v colour c , we can extend the colouring to the subtrees of u_1, \dots, u_j . If $|L_j(v)| > d - j$, then v is “non-critical”: we will always be able to assign it a colour after colouring the subtrees of u_{j+1}, \dots, u_d . This allows us to maintain that $|L_j(v)| \leq d - j$ and so the number of bits required to store the position of c in $L_j(v)$ decreases as j increases.

We need to be a bit more clever when we define the lists. The second main idea is to distribute the children of v into about $\log \log(n/2)$ brackets, where n denotes the number of vertices in the subtree below v . The distribution is done based on how much smaller the subtree of the child is compared to n . We allocate a specific number of bits per bracket: to brackets which allow bigger subtrees, we allocate less space. When “processing” brackets of smaller subtrees, we may need to store information as well for the brackets of bigger subtrees, but vice versa is not allowed. We choose the bracket sizes so that if we store information for the first j brackets, this “fits” in the space allocated to the $(j + 1)$ th bracket. In the end, the final algorithm is rather subtle and requires a careful analysis.

We outline some relevant definitions and background in Section 2. We give the simpler $O(\log^2 n)$ space algorithm in Section 3 and discuss the logspace algorithm in Section 4. We prove our results concerning tree-partition-width in Section 5 and point to some directions for future work in Section 6. Some technical details can be found in the full version [3].

2 Preliminaries

All logarithms in this paper have base 2. Let T be a rooted tree and $v \in V(T)$. We write T_v for the subtree rooted at v and $T - v$ for the forest obtained by removing v and all edges incident to v . We refer the reader to textbooks for basic notions in graph theory [9] and (parameterised) complexity [2, 12].

2.1 Simple logspace computations on trees

We repeatedly use the fact that simple computations can be done on a rooted tree using logarithmic space, such as counting the number of vertices in a subtree. We include a brief sketch below and refer to [22] for further details.

We first explain how to traverse a tree in logspace. Record the index of the current vertex and create states **down**, **next** and **up**. We start on the root with state **down**. When in state **down**, we go to the first child while remaining in the state **down**. If there is no child, we change the state to **next**. When in state **next**, we go to the next sibling if it exists and change state to **down**, or (if there is no next sibling) we change state to **up**. When in state **up**, we simultaneously go to the parent and change the state to **next**. We stop when reaching the root with state **up**. By keeping track of the number of vertices discovered, we can use the same technique to count the number of vertices in a subtree. This can then be used to compute the child with maximum subtree size and to enumerate children ordered by their subtree size. If the input tree is not rooted, we may use the indices of the vertices to root the tree in a deterministic way.

2.2 Graph width measures

Let $G = (V, E)$ be a graph. A tuple $(T, \{X_t\}_{t \in V(T)})$ is a *tree decomposition* for G if T is a tree, for $t \in V(T)$, $X_t \subseteq V$ is the *bag* of t , for each edge $uv \in E$ there is a bag such that $\{u, v\} \subseteq X_t$, and for each $u \in V$, the bags containing u form a nonempty subtree of T . If T is a path, this defines a *path decomposition*.

The width of such a decomposition is $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* (resp. *pathwidth*) of G is the minimum possible width of a tree decomposition (resp. path decomposition) of G .

A *nice* path decomposition has empty bags on endpoints of the path and two consecutive bags differ by at most one vertex. Hence, either a vertex is *introduced*, or a vertex is *forgotten*.

Let G be a graph, let T be a tree, and, for all $t \in V(T)$, let X_t be a non-empty set so that $(X_t)_{t \in V(T)}$ partitions $V(G)$. The pair $(T, (X_t)_{t \in V(T)})$ is a *tree-partition* of G if, for every edge $vv' \in E(G)$, either v and v' are part of the same bag, or $v \in X_t$ and $v' \in X_{t'}$ for $tt' \in E(T)$. The width of the partition is $\max_{t \in V(T)} |X_t|$. The *tree-partition-width* (also known as strong treewidth) of G is the minimum width of all tree-partitions of G . It was introduced by Seese [26] and can be characterised by forbidden topological minors [11]. Tree-partition-width is comparable to treewidth on graphs of maximum degree Δ [10, 27]: $\text{tw} + 1 \leq 2 \text{tpw} \leq O(\Delta \text{tw})$. However, it is incomparable to treedepth, pathwidth and treewidth for general graphs.

The *treedepth* of a graph is the minimum height of a forest F with the property that every edge of G connects a pair of nodes that have an ancestor-descendant relationship to each other in F .

3 Warm up: first ideas and a simpler algorithm

3.1 Storing colours via their position in the list

It is not too difficult to obtain a non-deterministic algorithm that uses $O(\log^2 n)$ space.

► **Proposition 3.** *LIST COLOURING can be solved non-deterministically using $O(\log n \log \Delta)$ space on n -vertex trees of maximum degree Δ .*

The proposition follows from the following two lemmas.

► **Lemma 4.** *LIST COLOURING can be solved non-deterministically using $O(k \log \Delta + \log n)$ space for an n -vertex graph G of maximum degree Δ if we can deterministically compute a path decomposition for G of width k in $O(\log n)$ space.*

A deterministic logspace algorithm for computing an optimal path decomposition exists for all graphs of bounded pathwidth [20], but this does not apply directly to trees (since their pathwidth may grow with n).

► **Lemma 5.** *If T is an n -vertex tree, we can deterministically construct a nice path decomposition of width $O(\log n)$ using $O(\log n)$ space.*

We remark that Δ may be replaced by a bound on the list sizes in Proposition 3 and Lemma 4. The main observation in the proof of Lemma 4 is that for a vertex v , we only need to consider the first $d(v) + 1$ colours from its list so that we can store the *position* of the colour rather than the colour itself. Note that we cannot keep the path decomposition in memory, but rather recompute it whenever any information is needed. We keep in memory only the current position in the path decomposition and the list positions of the colours we assigned for vertices in the previous bag.

3.2 Heavy children, recursive analysis and criticality

Suppose that we are given an instance (T, L) of LIST COLOURING. We fix a root v^* of T in an arbitrary but deterministic fashion, for example the first vertex in the natural order on the vertices. Let $v \in V(T)$. We see v as a descendant and ancestor of itself. We write T_v for the subtree with root v .

► **Definition 6** (Heavy). *A child u of a vertex v in a rooted tree T is called heavy if $|V(T_u)| \geq |V(T_{u'})|$ for all children u' of v , with strict inequality whenever $u' < u$ in the natural order on V .*

Each vertex has at most one heavy child. We also record the following nice property.

► **Observation 7.** *If u is a child of v which is non-heavy, then $|V(T_u)| \leq (|V(T_v)| - 1)/2$.*

An obvious recursive approach is to loop over the possible colour $c \in L(r)$ for the root r and then to recursively check for all children v of r whether a list colouring can be extended to the subtree T_v (while not giving v the colour c). We wish to prove a space upper bound of the form $S(n) = f(n) \log n$ on the number of bits of storage required for trees on n vertices (for some non-decreasing function f). We compute

$$S(n/2) = \log(n/2)f(n/2) \leq \log(n/2)f(n) = \log n f(n) - \log 2 f(n) \leq S(n) - f(n). \quad (1)$$

This shows that while performing a recursive call on some subtree T_v with $|V(T_v)| = n/2$, we may keep an additional $f(n)$ bits in memory (on top of the space required in the recursive call). In particular, we can store the colour c using $O(\log n)$ bits when $f(n) = \Theta(\log n)$, but can only keep a *constant* number of bits for such recursions when proving Theorem 1.

We next explain how we can ensure that we only need to consider recursions done on non-heavy children. Suppose v has non-heavy children v_1, \dots, v_k and heavy child u . We will write $G_v = T_v - T_u$. Suppose the parent v' of v needs to be assigned colour c' . One of the following must be true.

- There is no colouring of G_v which avoids colour c' for v . In this case, we can reject.
- There is a unique colour $c \neq c'$ that can be assigned to v in a list colouring of G_v . We say v is *critical* and places the colour constraint on u that it cannot receive colour c .

- There are two possible colours unequal to c' that we can give v in G_v . We then say v is *non-critical*: it can be coloured for each colour we might wish to assign its heavy child u . One example of non-criticality is if the list of available colours $|L(v)| \geq k + 2$. In this case, T_v is list colourable if and only if $T_{v_1}, \dots, T_{v_k}, T_u$ are all list colourable (using lists L).

3.3 A deterministic algorithm using $O(\log^2 n)$ space and polynomial time

We define a procedure $\text{SOLVE}(v, p)$ which given a vertex v and number $p \in \{0, 1, \dots, n\}$, determines whether the subtree T_v rooted on v can be list coloured; for $p \geq 1$ there is an additional constraint that v cannot receive the p th colour in $L(v)$; for $p = 0$ the vertex v may receive any colour.

Suppose we call $\text{SOLVE}(r, p)$ for some $r \in V(T)$. Let v_1, \dots, v_k denote the non-heavy children of r and u the heavy child. The algorithm works as follows.

1. For $i = 1, \dots, k$, we recursively verify that T_{v_i} can be list coloured ($\text{SOLVE}(v_i, 0)$); we reject if any of these rejects.
2. If $|L(r)| \geq d(r) + 1 = k + 2$, then we are “non-critical”: we free up our memory (removing also (r, p)) and recursively verify whether T_u can be list coloured by calling $\text{SOLVE}(u, 0)$.
3. From now on, $|L(r)| \leq k + 1$. We check that there is some $p_1 \in \{1, \dots, |L(r)|\}$ for which we can assign v the p_1 th colour in its list, and extend to a list colouring of $T \setminus T_u$. This involves recursive calls $\text{SOLVE}(v_i, p_{i,1})$ where $p_{i,1}$ places the appropriate constraint² on v_i . If no such p_1 exists, we reject.
4. Next, we check whether r is “non-critical”, that is, whether there is some $p_2 \neq p_1$ for which there is a list colouring of $T \setminus T_u$ in which r receives the p_2 th colour from its list. If such p_2 exists, we free up our memory and recursively verify whether T_u can be list coloured. If p_2 does not exist, then we know that r must get colour p_1 . If $p_1 = p$, we reject. Otherwise, we free up our memory and run $\text{SOLVE}(u, p'_1)$, where p'_1 is either 0 or the position in the list of u of the p_1 th colour of $L(r)$.

We give a brief sketch of the space complexity; more precise arguments including also pseudocode and the time analysis of this algorithm are given in the full version [3].

For the sake of analysis, suppose we keep a counter \mathbf{r} that keeps track of the “recursion depth”. We increase this by one each time we do a call on a non-heavy child (decreasing it again once it finishes), but do not adjust it for calls on a heavy child.

Suppose \mathbf{r} has been increased due to a sequence of recursive calls on $(v_1, 0), (v_2, p_2), \dots, (v_\ell, p_\ell)$, with v_1 the root of the tree and v_{i+1} a non-heavy child of some heavy descendant of v_i for all $i \in [\ell - 1]$. Then $|V(T_{v_\ell})| \leq \frac{1}{2}|V(T_{v_{\ell-1}})| \leq \dots \leq \frac{1}{2^{\ell-1}}|V(T)|$. In particular, \mathbf{r} is always bounded by $\log n$.

Crucially, if a call $\text{SOLVE}(u, p')$ is made on the heavy descendant u of some v_i , the only information we need to store relating to the part of the tree “between v_i and u ” is p' . Therefore, if we distribute our work tape into $\lceil \log n \rceil$ parts where the i th part will be used whenever \mathbf{r} takes the value i , then each part only needs to use $10 \log n$ bits, giving a total space complexity of $O(\log^2 n)$.

² Let c_1 be the p_1 th colour in $L(r)$. If $c_1 \notin L(v_i)$, let $p_{i,1} = 0$. Otherwise, let $p_{i,1}$ be the position of c_1 in $L(v_i)$.

4 Proof of Theorem 1

In this section, we describe our $O(\log n)$ space algorithm. This also uses the ideas of using positions in a list (rather than the colours themselves), criticality and starting with the non-heavy children described in the previous section. However, we need to take the idea of first processing “less heavy” children even further.

The main idea is to store the colour c that we are trying for a vertex v using the position p_j of c in some list $L_j(v)$, and to reduce the size of the list (and therefore the storage requirement of p_j) before we process “heavier” children of v . There are two key elements:

- We can recompute c from p_j in $O(\log n)$ space. This is useful, since we can do a recursive call while only having p_j (rather than c) as overhead, discover some information, and then recompute c only at a point where we have a lot of memory available to us again.
- The space used for p_j will depend on the size of the tree that we process. It is too expensive to consider all sizes separately, and therefore we will “bracket” the sizes. Subtrees whose size falls within the same bracket are processed in arbitrary order. For example, we put all trees of size $O(\sqrt{n})$ in a single bracket: these can be processed while using $O(\log n)$ bits of information about c (which is trivially possible). At some point we reach brackets for which the subtrees have size linear in n , say of size at least $\frac{1}{4}n$. Then, we may only keep a *constant* number of bits of information about c . Intuitively, this is possible because at most four children of r can have a subtree of size $\geq \frac{1}{4}n$, so the “remaining degree” of v is small. In particular, if more than six colours for v work for all smaller subtrees, then v is “non-critical”.

We first explain our brackets in Section 4.1. We then explain how we may point to a colour using less memory in Section 4.2 and how we keep track of vertices using less memory in Section 4.3. We then sketch the proof of Theorem 1 by outlining the algorithm and its space analysis in Section 4.4.

4.1 Brackets

Recall that we fixed a root for T in an arbitrary but deterministic fashion.

Let $v \in V(T)$ and u the heavy child of v . Let $G_v = T_v - T_u$ and $n_v = |V(T_v)|$. Each subtree T' of $G_v - v$ is rooted in a non-heavy child of v and will be associated to a bracket based on $|V(T')|$. By Observation 7, $1 \leq |V(T')| \leq (n_v - 1)/2$. Let $M_v = \lceil \log \log(n_v/2) \rceil$. The brackets are given by the sets of integers in the intervals

$$[1, n_v/2^{2^{M_v-1}}), [n_v/2^{2^{M_v-1}}, n_v/2^{2^{M_v-2}}), \dots, [n_v/256, n_v/16), [n_v/16, n_v/4), [n_v/4, n_v/2).$$

There are M_v brackets: $[n_v/2^{2^j}, n_v/2^{2^{j-1}})$ is the j th bracket for $j \in \{1, \dots, M_v - 1\}$ and $[1, n_v/2^{2^{M_v-1}})$ is the M_v th bracket. Note that $n_v/2^{2^{M_v-1}} = O(\sqrt{n_v})$. This implies that while doing a recursive call on a tree in the M_v th bracket, we are happy to keep an additional $O(\log n_v)$ bits in memory.

We aim to show that for some universal constant C , when doing a recursive call on a tree in the j th bracket, we can save all counters relevant to the current call using at most $C2^j$ bits (which depending on the value of j , could be $\Omega(\log n)$). In our analysis, we save the counters in a new read-only part of the work tape. The recursive call cannot alter this (and will have to work with less space on the work space). We can then use our saved state to continue with our calculations once the recursive call finishes.

We short-cut $M = M_v$ for legibility; the dependence of M on v is only needed to ensure that we do not start storing counters for lots of empty brackets when n_v is much smaller than n , and can be mostly ignored.

4.2 The information p_j stored about a colour c

Let $v \in V(T)$ with heavy child u and $c \in L(v)$. Recall that $G_v = T_v - T_u$. We will loop over $j = M, \dots, 0$ and consider subtrees of G_v whose size falls in the j th bracket in an arbitrary, but deterministic order (e.g. using the natural order on their roots). When j decreases, we will perform recursions on larger subtrees of $G_v - v$ and can therefore keep less information about c . We first define “implicit” lists.

- Set $L_M(v) = L(v)$.
- For $j \in [0, M - 1]$, let $L_j(v)$ be the set of colours $\alpha \in L(v)$ such that all subtrees T' of $G_v - v$ with $|V(T')| < n/2^{2^j}$ can be coloured without giving the colour α to the root of T' (so that v may receive colour α according to those subtrees).

Note that $L_j(v) = L(v)$ if there are no subtrees T' with $|V(T')| < n/2^{2^j}$. Since the subtrees associated to brackets $1, \dots, j$ have size at least $n/2^{2^j}$, there can be at most 2^{2^j} of them. If $|L_j(v)| \geq 2^{2^j} + 3$ and all subtrees T' of $G_v - v$ can be coloured, then v is “non-critical”: after the parent and heavy child of v have been coloured, the colouring can always be extended to v and the rest of G_v .

Suppose we are testing if we can give colour c to v . If $c \notin L_j(v)$, then we may reject: c is not a good colour for one of the subtrees. We define $p_j = p_j(c, v)$ as the integer $x \in \{1, \dots, |L_j(v)|\}$ such that the x th element in $L_j(v)$ equals c . In particular, p_M is the position of the colour c in the list $L_M(v) = L(v)$ and p_0 gives the position of c in the list of colours for which all subtrees of G_v allow v to receive c . For $j < M$, we will reserve at least $\log(2^{2^j} + 3)$ bits for p_j . This is possible, because we can maintain that $|L_j(v)| \leq 2^{2^j} + 2$ by going into a “non-critical subroutine” if we discover the list is larger.

4.3 Position of the current vertex

Next, we describe how to obtain efficient descriptions of the vertices in the tree. When performing recursions, we find it convenient to store information using which we can retrieve the “current vertex” of the parent call. Therefore, we require small descriptions for such vertices if the call did not make much “progress”.

For any $v \in V(T)$, define a sequence $h(v, 1), h(v, 2), \dots$ of heavy descendants as follows. Let $h(v, 1) = v$. Having defined $h(v, i)$ for some $i \geq 1$, if this is not a leaf, we let $h(v, i + 1)$ be the heavy child of $h(v, i)$. Note that given the vertex $h(v, i)$, we can find the vertex $h(v, i + 1)$ (or conclude it does not exist) in $O(\log n)$ space. We give a deterministic way of determining a bit string $\text{pos}(v, i)$ that represents $h(v, i)$, where the size of the bit string will depend on the “progress” made at the vertex $h(v, i)$ that it represents. This “progress” is measured by the size t_i of the largest subtree T' of a non-heavy child of $h(v, i)$, that is, T' is the largest component of $T - h(v, i)$ which does not contain $h(v, j)$ for $j \neq i$. We define $\text{pos}(v, i)$ as follows.

- Let j be given such that $t_i \in [n_v/2^{2^j}, n_v/2^{2^{j-1}})$. Start $\text{pos}(v, i)$ with j zeros, followed by a 1.
- There are at most 2^{2^j} values of i for which $t_i \geq 2^{2^j}$. We add a bit string of length 2^j to $\text{pos}(v, i)$, e.g. the value x for which $h(v, i)$ is the x th among $h(v, 1), \dots, h(v, a)$ with $t_i \in [n_v/2^{2^j}, n_v/2^{2^{j-1}})$.

Note that, given v , we can compute $\text{pos}(v, i)$ from $h(v, i)$ and $h(v, i)$ from $\text{pos}(v, i)$ using $O(\log n)$ space. If we do a recursive call, it will be on a non-heavy child u of some $h(v, i)$. By definition, $|V(T_u)| \leq t_i$ and $\text{pos}(v, i)$ depends on t_i in a way that we are able to keep it in memory while doing the recursive call.

We use the encoding $(\text{pos}(v^*, i), j, \ell)$ for the ℓ th child u of $h(v^*, i)$ whose subtree has a size that falls in the j th bracket. We can also attach another such encoding, e.g.

$$((\text{pos}(v^*, i), j, \ell), (\text{pos}(u, i'), j', \ell')),$$

to keep track of u and the ℓ' th child v of $h(u, i')$ whose subtree has a size that falls in the j' th bracket. We can retrieve v from the encoding above in $O(\log n)$ space and therefore can retrieve it whenever we have such space available to us.

4.4 Description of the algorithm

During the algorithm, the work tape will always start with the following.

- \mathbf{r} : the recursion depth r written in unary. At the start, $r = 0$.
- $\mathbf{pos} = \text{pos}_0 | \dots | \text{pos}_r$: encodes vertices as described Section 4.3. At the start, this is empty and points at the root v^* of the tree on the input tape.
- $\mathbf{p} = \mathbf{p}_0 | \dots | \mathbf{p}_r$: encodes colour restriction information for the vertices encoded by \mathbf{pos} . At the start, this is empty and no restrictions are given. We maintain throughout the algorithm that \mathbf{p}_i gives us colour restrictions for the vertex v pointed at by pos_i . The restriction can either be “no restrictions” or “avoid c' ”; in the latter case \mathbf{p}_i contains a tuple (j, p'_j) with p'_j the position of c' in $L_j(v')$, where v' is the parent of v .
- $\mathbf{aux} = \mathbf{aux}_0 | \dots | \mathbf{aux}_r$: further auxiliary information for parent calls that may not be overwritten.

We define a procedure **process**. While the value of \mathbf{r} equals r , the bits allocated to $\text{pos}_i, \mathbf{p}_i, \mathbf{aux}_i$ for $i < r$ will be seen as part of the read-only input-tape. In particular, the algorithm will not make any changes to $\text{pos}_i, \mathbf{p}_i$ or \mathbf{aux}_i for any $i < r$, but may change the values for $i = r$.

We will only increase \mathbf{r} when we do a recursive call. If during the run of the algorithm $\mathbf{r} = r$ and a recursive call is placed, then we increase \mathbf{r} to $r + 1$ and return to the start of our instructions. However, since \mathbf{r} has increased it will now see a “different input tape”. When the call finished, we will decrease \mathbf{r} back to r and wipe everything from the work space except for $\mathbf{r}, \text{pos}_i, \mathbf{p}_i, \mathbf{aux}_i$ for $i \leq r$, and the answer of the recursive call (0 or 1). We then use \mathbf{aux}_i to reset our work space and continue our calculations.

We will ensure that \mathbf{r} is always upper bounded by $\log n$. Indeed, the vertex v_r encoded by pos_r will always be a non-heavy child of a descendant of the vertex encoded by pos_{r-1} .

While $\mathbf{r} = r$, the algorithm is currently doing calculations to determine whether the vertex v pointed at by pos_{r-1} (v^* for $r = 0$) has the property that T_v can be list coloured, while respecting the colour restrictions encoded by \mathbf{p}_{r-1} (none if $r = 0$). Recall that rather than writing down v explicitly, we use a special encoding from which we can recompute v whenever we have $C \log n$ space available on the work tape (for some universal constant C). Similarly, \mathbf{p}_{r-1} may give a position p'_j in $L_j(v')$ for some $j < M$ (for v' the parent of v), and we may need to use our current work tape to recompute the corresponding position p'_M of the colour in $L(v')$, so that we can access the colour from the input tape. This part will make the whole analysis significantly more technical.

We define an algorithm which we call **process** as follows. A detailed outline is given in the full version [3], whereas an informal description of the steps is given below.

0. Let v be the vertex pointed at by pos_r . We maintain that at most one colour c' has been encoded that v must avoid. Handle the case in which v is a leaf. If not, it has some heavy child h . We go to 1, which will eventually lead us to one of the following (recall that $G_v = T_v - T_h$):

(rej) There is no list colouring of G_v avoiding c' for v . We return **false**.

- (nc) The vertex v can get two colours (unequal to c') in list colourings of G_v . In this case, we say v is non-critical. We update pos_r to h , set p_r to “none” and repeat from 0.
- (cr) There is a unique colour $c \neq c'$ that works. Then $\{c\} \subseteq L_0(v) \subseteq \{c, c'\}$ and so can represent $p_0 = p_0(c, v)$ with a single bit. We update pos_r to h , update p_r to $(0, p_0)$ and repeat from 0.
1. We check that all subtrees can be coloured if we do not have any colour restrictions (necessary for the non-critical subroutine). This involves recursive calls on **process** where we have no colour constraint on the root of the subtree.
 2. We verify that $L_0(v)$ is non-empty. We iteratively try to compute p_M from $p_0 = 1$ via p_1, \dots, p_{M-1} . Starting from $p_0 = 1$ and $j = 0$, we compute p_{j+1} from p_j as follows.
 - (i) Initialise $\text{curr}_j = 1$. This represents a position in $L_j(v)$ (giving the number of “successes”).
Initialise $\text{prev}_j = 1$. This represents a position in $L_{j+1}(v)$ (giving the number of “tries”).
 - (ii) We check whether the prev_j th colour of $L_{j+1}(v)$ works for the trees in the j th bracket. This involves a recursive call on **process** for each tree T' in the j th bracket, putting $(j + 1, \text{prev}_j)$ as the colour constraint for the root of T' . (The colour restriction gives a position in $L_{j+1}(v)$; we do not store the corresponding colour and rather will recompute it in the recursive call!)
 - (iii) If one of those runs fails, we increase prev_j ; if this is now $> 2^{2^{j+1}} + 3$, then this implies a lower bound on $|L_{j+1}(v)|$ which allows us to move to (nc).
If $\text{curr}_j < p_j$ we increase both curr_j and prev_j .
Once $\text{curr}_j = p_j$, we have successfully computed $p_{j+1} = \text{prev}_j$ and continue to compute p_{j+2} if $j + 1 < M$. Otherwise we repeat from (ii).

Once we have p_M , we can access the corresponding colour from the input tape by looking at the p_M th colour of $L(v)$. If $p_M > |L(v)|$, we go to (rej).

3. We verified $|L_0(v)| \geq 1$. We establish whether $|L_0(v)| \geq 3$ in a similar manner. If so, we can go to (nc); else we need to start considering the colour constraints of the parent v' of v . Note that we can use aux_r to store auxiliaries such as $\alpha = |L_0(v)| \in \{1, 2\}$.

It remains to explain how we check whether the first or second colour from $L_0(v)$ satisfies the colour constraint from v' . Suppose a colour c' has been encoded via the position $p'_{j'}$ of c' in $L_j(v')$ for some $j' \in [0, M']$ (where $M' = M_{v'}$).

We can recompute $p'_{M'}$ from $p'_{j'}$ in a similar manner to the above. However, once we store $p'_{M'}$, we may no longer be able to compute p_M from $p_0 = 1$ or $p_0 = 2$, since $p'_{M'}$ may take too much space³. Therefore, we first recompute $p_{j'}$ from p_0 and then simultaneously recompute $p_{j'+1}$ from $p_{j'}$ and $p'_{j'+1}$ from $p'_{j'}$ until we computed p_M and $p'_{M'}$. We then check whether the p_M th colour of $L(v)$ equals c' , the $p'_{M'}$ th colour of $L(v')$. (It may be that $M \neq M'$, meaning that we may finish one before the other.)

The computation of p'_{x+1} from p'_x for the parent v' of v is a bit more complicated if v is a non-heavy child of v' . In this case, v is in the $(j' - 1)$ th bracket of v' . The algorithm calls again on itself for subtrees in the x th bracket of v' , but now we see a resulting call to **process** as a *same-depth call* rather than a “recursive call”. The computations work the same way, but we do not adjust r and will *add* the current state from before the call in aux_r .

³ This is one of the issues that made this write-up more technical and involved than one might expect necessary at first sight; if we do a recursion on a child in the j th bracket of v or v' , then we are only allowed to keep $O(2^j)$ bits on top of the space used by the recursion. This means we cannot simply keep $p'_{M'}$ in memory if j is much smaller than M' .

By an exhaustive case analysis, the algorithm computes the right answer. It remains to argue that it terminates and runs in the correct space complexity.

We first further explain the same-depth calls on `process`. When we make such call, we do not adjust \mathbf{r} . When the call is made, \mathbf{pos}_r will encode a vertex v_1 which is a non-heavy child of v' . After the call, \mathbf{pos}_r will point to some other non-heavy child v_2 of v' . Importantly, the bracket of v_2 will always be higher than the bracket of v_1 , so that at most M such same-depth calls are made before changing our recursion level. Each same-depth call may stack on a number of auxiliary counters which we keep track of using \mathbf{aux}_r ; since for each j , there is at most one vertex from the j th bracket which may append something to this, it suffices to ensure a vertex from bracket j adds at most $C2^j$ bits. Indeed, this ensures that the size of \mathbf{aux}_r takes at most $C \sum_{j=1}^{j'} 2^j \leq C2^{j'+1} = O(2^{j'})$ bits once \mathbf{pos}_r encodes a vertex from bracket j' .

We now argue that it terminates. Each time we do a same depth call, the value of the bracket j will increase by at least one. We therefore may only do a finite number of same-depth calls in a row. When $\mathbf{r} = 0$, each time we reach (nc) or (cr), we move one step on the heavy path from the root to a leaf, so this will eventually terminate. A similar observation holds for $\mathbf{r} > 0$ once we fix $\mathbf{pos}_1 | \dots | \mathbf{pos}_{r-1}$: this points to some vertex v and \mathbf{pos}_r will initially point to a non-heavy descendent u or v , and then “travel down the heavy path” from u to a leaf.

We next consider the space used by the algorithm. Let $S(n)$ be the largest amount of bits used for storing \mathbf{r} , \mathbf{pos} , \mathbf{p} and \mathbf{aux} during a run of `PROCESS` on an n -vertex tree. We can distribute our work space into two parts: $C_1 \log n$ space for temporary counters and for doing calculations such as computing the heavy child of a vertex, and $S(n)$ bits for storing \mathbf{r} , \mathbf{pos} , \mathbf{p} and \mathbf{aux} . It suffices to prove that $S(n) \leq C_2 \log n$; this is the way we decided to formalise keeping track of the “overhead” caused by recursive calls.

Note that \mathbf{r} is bounded by $\log n$ if the input tree has n vertices: each time it increases, we moved to a non-heavy child whose subtree consists of at most $n/2$ vertices.

Suppose we call `process` with \mathbf{pos} pointing at some vertex v whose subtree has size n_v . We will show inductively that the number of bits used by \mathbf{pos} , \mathbf{p} and \mathbf{aux} is in $O(\log n_v)$ throughout this call (note that n_v may be much smaller than n , the number of vertices of the tree on the input tape). Whenever we do a recursive call, this will be done on a tree whose size is upper bounded by $n_v/2^{2^{j-1}}$ for some $1 \leq j \leq M = \lceil \log \log(n_v/2) \rceil$. Since by induction the recursive call requires only

$$S(n/2^{2^{j-1}}) = C_2 \log(n/2^{2^{j-1}}) \leq S(n) - C_2 2^{j-1}$$

additional bits, we will allow ourselves to add at most $C_2 2^{j-1}$ bits to \mathbf{r} , \mathbf{pos} , \mathbf{p} and \mathbf{aux} before we do a recursive call that divides the number of vertices by at least $2^{2^{j-1}}$. The constant C_2 will be relatively small (for example, 1000 works).

Fix a value of j . From the definitions in Section 4.2 and 4.3, at the point that we do a recursion on a subtree whose size is upper bounded by $n_v/2^{2^{j-1}}$, \mathbf{pos}_r and \mathbf{p}_r store at most three integers (they are of the form $(\mathbf{pos}(v', i), j, \ell)$ and (x, p_x) respectively) that are bounded by $2^{2^j} + 3$. Therefore, these require at most $C' 2^{j-1}$ bits for some constant C' (e.g. $C' = 50$ works).

The worst case comes from \mathbf{aux}_r which may get stacked up on by the “same-depth calls”. There is at most one such same-depth call per $x \in \{1, \dots, j\}$. For such x , we add on a bounded number of counters (e.g. \mathbf{curr}_x and \mathbf{prev}_x) which can take at most $2^{2^{x+1}} + 3 \leq 2 \cdot 2^{2^{x+1}}$ values. Since $C \sum_{x=1}^j 2^x \leq 4C2^{j-1}$, \mathbf{aux}_r also never contains more than $O(2^{j-1})$ bits while doing a call on a tree whose size falls in bracket j .

5 Graphs of bounded tree-partition-width

5.1 Proof of Corollary 2

Here we sketch the proof of Corollary 2. Let $(T, (X_t)_{t \in V(T)})$ be a tree-partition of width k for an N -vertex graph G , where it will be convenient to write $n = |V(T)| \leq N$. We prove that there is an algorithm running in $O(k \log k \log n)$ space, for each k by induction on n .

We root T in the vertex of lowest index. We define the recursion depth, heavy children, the brackets, M and “position” for the vertices in T exactly as we did in Section 4.

Suppose pos points at some vertex $t \in V(T)$. We aim to use $O(k \log k \log(2^{2^j}))$ bits for the information p_j stored about the colouring c of the vertices in X_t while processing subtrees in bracket j . This is done as follows. For $t \in V(T)$, let $G_{t,j}$ be the graph induced on the vertices that are either in X_t or in X_s for s a non-heavy child of t in bracket j or above. For $v \in X_t$, we define $L_j(v)$ to be the list of colours $\alpha \in L(v)$ such that there is a list colouring of $G_{t,j}$ that assigns the colour α to v . There are at most 2^{2^j} subtrees of T associated to brackets $1, \dots, j$, and so these include at most $k2^{2^j}$ neighbours of v . Therefore, if $|L_j(v)| \geq k(2^{2^j} + 3)$, then G can be list coloured if and only if $G - v$ can be list coloured and we no longer need to keep track of the colour of v . We then establish v is non-critical. We use k bits to write for each vertex of X_t whether or not it has been established to be critical, and for those vertices that are critical, we use $\log(k(2^{2^j} + 3))$ bits per vertex to index a colour from $L_j(v)$.

We use $k + \log((3k)^k) = O(k \log k)$ bits of information in aux to keep track of the following.

- For each $v \in X_t$, whether or not v has been established to be critical. After processing t , the vertex $v \in X_t$ will be critical if there are at most $3k$ colours in $L(v)$ for which there exists an extension to G_t . Let $C_t \subseteq X_t$ denote the critical vertices.
- For each $p_0 \in \prod_{v \in C_t} L_0(v)$ (of which there are at most $(3k)^k$), a single bit which indicates whether or not the parent of t would allow the corresponding colouring.

While computing the information above, we still need the auxiliary information from the parent of t , but we can discard this by the point we start processing the heavy child of t .

We make two more small remarks:

- We need to redefine what we mean by “increasing” p_j for some j , since we now work with a tuple of list positions. We fix an arbitrary but deterministic way to do this, for example in lexicographical order using the natural orders on the vertices and colours.
- When we check whether the colouring c of X_t corresponding to $p_0 \in \prod_{v \in C_t} L_0(v)$ is allowed by the parent t' of t , we run over $p'_0 \in \prod_{v' \in C_{t'}} L_0(v')$, and as before we need to compute p_i, p'_i from p_{i-1}, p'_{i-1} iteratively until we obtain the colourings corresponding to p_M and p'_M . If those colourings are compatible, then we know that there is a list colouring of the graph “above t ” for which X_t is coloured “according to p_0 ”, and so we record in aux that p_0 is allowed.

The calculations in the space analysis work in the exact same way: we simply multiply everything by $Ck \log k$ (for a universal constant C).

5.2 W[1]-hardness

We give an easy reduction for the following result.

► **Theorem 8.** *LIST COLOURING parameterised by the width of a given tree-partition is W[1]-hard.*

Proof. We reduce from MULTICOLOURED CLIQUE.

Consider a MULTICOLOURED CLIQUE instance $G = (V, E), V_1, \dots, V_k$ with $k \geq 2$ colours. We denote by $\overline{G} = (V, \overline{E})$ the complement of G .

We now describe the construction of our instance graph H . We first add vertices v_1, \dots, v_k , with lists $L(v_i) = V_i$ for all $i \in [k]$. Then for each edge $e = uv \in \overline{E}$ we add a vertex x_{uv} with list $L(x_{uv}) = \{u, v\}$. Furthermore, for $\alpha \in \{u, v\}$, let i be such that $\alpha \in V_i$. We add the edge $x_{uv}v_i$.

The resulting graph has tree-partition-width at most k : we put v_1, \dots, v_k in the same bag which is placed at the centre of a star, and create a separate leaf bag containing x_{uv} for each $uv \in \overline{E}$.

▷ **Claim 9.** If there is a proper list colouring of H , then there is a multicoloured clique in G .

Proof. Suppose that H admits a list colouring. Let $a_i \in V_i = L(v_i)$ be the colour assigned to v_i for all $i \in [k]$. We will prove a_1, \dots, a_k forms a multicoloured clique in G .

Consider distinct $i, j \in [k]$ and suppose $a_i a_j$ is not an edge of G , that is, $a_i a_j \in \overline{E}$. Then there exists a vertex $x_{a_i a_j}$ adjacent to both v_i and v_j , but there is no way to properly colour it, a contradiction. So we must have $a_i a_j \in E$ as desired. ◁

▷ **Claim 10.** If there is a multicoloured clique in G , then there is a proper list colouring of H .

Proof. We denote by a_1, \dots, a_k the vertices of the multicoloured clique, where $a_i \in V_i$ for all i . We assign the colour a_i to vertex v_i . Consider now x_{uv} for some $uv \in \overline{E}$. Let i and j be given so that x_{uv} is adjacent to v_i and v_j . Then $\{u, v\} \neq \{a_i, a_j\}$ since $a_i a_j \in E$ and $uv \in \overline{E}$. Therefore, we may assign either u or v (or both) as colour to x_{uv} . ◁

Since MULTICOLOURED CLIQUE is W[1]-hard, this proves that LIST COLOURING parameterised by tree-partition-width is W[1]-hard. ◀

We remark that the above proof also shows that LIST COLOURING parameterised by vertex cover is W[1]-hard.

6 Conclusion

In this paper, we combined combinatorial insights and algorithmic tricks to give a space-efficient colouring algorithm.

By combining Logspace Bodlaender's theorem [14], Lemma 5 and Lemma 4, LIST COLOURING can be solved non-deterministically on graphs of pathwidth k in $O(k \log n)$ space and on graphs of treewidth k in $O(k \log^2 n)$ space.⁴ However, we already do not know the answer to the following question.

▶ **Problem 11.** *Can a non-deterministic Turing machine solve LIST COLOURING for n -vertex graphs of treewidth 2 using $o(\log^2 n)$ space?*

Another natural way to extend trees is by considering graphs of bounded treedepth. Such graphs then also have bounded pathwidth (but the reverse may be false). It has been observed for several problems such as 3-COLOURING and DOMINATING SET that “dynamic programming approaches” (common for pathwidth or treewidth) require space exponential in the width parameter, whereas there is a “branching approach” with space polynomial in

⁴ First compute a tree decomposition $(T, (B_t)_{t \in V(T)})$ of width k for G in $O(\log n)$ space [14], and then compute a (not necessarily optimal) path decomposition of width $O(\log |V(T)|)$ in $O(\log n)$ space for T , and turn this into a path decomposition for G of width $O(k \log n)$ by replacing $t \in V(T)$ with the vertices in its bag B_t . Then use Lemma 4.

treedepth [6]. A simple branching approach also allows LIST COLOURING to be solved in $O(k \log n)$ space on n -vertex graphs of treedepth k . We wonder if the approach in our paper can be adapted to improve this further.

► **Problem 12.** *Can LIST COLOURING be solved in $f(k)g(n) + O(\log n)$ space on graphs of treedepth k , with $g(n) = o(\log n)$ and f a computable function?*

Another interesting direction is what the correct complexity class is for LIST COLOURING parameterised by tree partition width. We do not expect this to be in the W-hierarchy because the required witness size seems to be too large. Moreover, the conjecture [23, Conjecture 2.1] mentioned in the introduction together with Corollary 2 would imply that the problem is not XNLP-hard.

Finally, it would be interesting to study other computational problems than LIST COLOURING. We remark that our results are highly unlikely to generalise to arbitrary Constraint Satisfaction Problems. Recall that there is conjectured to be a $k^* \in \mathbb{N}$ for which LIST COLOURING requires $\omega(\log n)$ space for n -vertex graphs of treewidth k^* . Since LIST COLOURING on n -vertex graphs of treewidth at most k^* can be reduced in logspace to a CSP on at most n variables, each having a lists of size at most n^{k^*} , and binary constraints on the variables, such CSP problems would then also require $\omega(\log n)$ space since k^* is a constant.

References

- 1 Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: Time–space tradeoffs. *Theory of Computing*, 10:297–339, 2014. doi:10.4086/toc.2014.v010a012.
- 2 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009. URL: <http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264>.
- 3 Hans L. Bodlaender, Carla Groenland, and Hugo Jacob. List colouring trees in logarithmic space. *arXiv*, CoRR(2206.09750), 2022. doi:10.48550/ARXIV.2206.09750.
- 4 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *Proceedings 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 193–204, 2021. doi:10.1109/FOCS52979.2021.00027.
- 5 Hans L. Bodlaender, Carla Groenland, and Céline M. F. Swennenhuis. Parameterized complexities of dominating and independent set reconfiguration. In Petr A. Golovach and Meirav Zehavi, editors, *Proceedings 16th International Symposium on Parameterized and Exact Computation, IPEC 2021*, volume 214 of *LIPICs*, pages 9:1–9:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.9.
- 6 Li-Hsuan Chen, Felix Reidl, Peter Rossmanith, and Fernando Sánchez Villaamil. Width, depth, and space: Tradeoffs between branching and dynamic programming. *Algorithms*, 11(7):98, 2018. doi:10.3390/a11070098.
- 7 Yijia Chen, Jörg Flum, and Martin Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *Proceedings 18th Annual IEEE Conference on Computational Complexity, CCC 2003*, pages 13–29. IEEE Computer Society, 2003. doi:10.1109/CCC.2003.1214407.
- 8 Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. In *Proceedings 24th Annual IEEE Conference on Computational Complexity, CCC 2009*, pages 203–214. IEEE Computer Society, 2009. doi:10.1109/CCC.2009.16.
- 9 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2012.
- 10 Guoli Ding and Bogdan Oporowski. Some results on tree decomposition of graphs. *J. Graph Theory*, 20(4):481–499, 1995. doi:10.1002/jgt.3190200412.

- 11 Guoli Ding and Bogdan Oporowski. On tree-partitions of graphs. *Discret. Math.*, 149(1-3):45–58, 1996. doi:10.1016/0012-365X(94)00337-I.
- 12 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.
- 13 László Egri, Pavol Hell, Benoît Larose, and Arash Rafiey. Space complexity of list H -colouring: A dichotomy. In Chandra Chekuri, editor, *Proceedings 25th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 349–365. SIAM, 2014. doi:10.1137/1.9781611973402.26.
- 14 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 143–152. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.21.
- 15 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.
- 16 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011. doi:10.1016/j.ic.2010.11.026.
- 17 Naveen Garg, Marina Papatriantafilou, and Philippas Tsigas. Distributed list coloring: How to dynamically allocate frequencies to mobile base stations. In *Proceedings 8th IEEE Symposium on Parallel and Distributed Processing, SPDP 1996*, pages 18–25. IEEE Computer Society, 1996. doi:10.1109/SPDP.1996.570312.
- 18 Sylvain Gravier, Daniel Kobler, and Wieslaw Kubiak. Complexity of list coloring problems with a fixed total number of colors. *Discrete Applied Mathematics*, 117(1-3):65–79, 2002.
- 19 Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75(2):135–155, 1997. doi:10.1016/S0166-218X(96)00085-6.
- 20 Shiva Kintali and Sinziana Munteanu. Computing bounded path decompositions in logspace. *Electron. Colloquium Comput. Complex. (ECCC)*, page 126, 2012. URL: <https://eccc.weizmann.ac.il/report/2012/126>.
- 21 Jan Kynčl and Tomáš Vyskočil. Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Trans. Comput. Theory*, 1(3), 2010. doi:10.1145/1714450.1714451.
- 22 Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings 24th Annual ACM Symposium on Theory of Computing, STOC 1992*, STOC '92, pages 400–404, New York, NY, USA, 1992. Association for Computing Machinery. doi:10.1145/129712.129750.
- 23 Michal Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.
- 24 Krishna N. Ramachandran, Elizabeth M. Belding-Royer, Kevin C. Almeroth, and Milind M. Buddhikot. Interference-aware channel assignment in multi-radio wireless mesh networks. In *Proceedings 25th IEEE International Conference on Computer Communications, INFOCOM 2006*, pages 1–12. IEEE Computer Society, 2006. doi:10.1109/INFOCOM.2006.177.
- 25 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008. doi:10.1145/1391289.1391291.
- 26 Detlef Seese. Tree-partite graphs and the complexity of algorithms. In Lothar Budach, editor, *Proceedings 5th International Conference on Fundamentals of Computation Theory, FCT 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1985. doi:10.1007/BFb0028825.
- 27 David R. Wood. On tree-partition-width. *Eur. J. Comb.*, 30(5):1245–1253, 2009. doi:10.1016/j.ejc.2008.11.010.

Dynamic Coloring of Unit Interval Graphs with Limited Recourse Budget

Bartłomiej Bosek  

Theoretical Computer Science Department, Faculty of Mathematics and Computer Science,
Jagiellonian University in Kraków, Poland

Anna Zych-Pawlewicz¹  

University of Warsaw, Poland

Abstract

In this paper we study the problem of coloring a unit interval graph which changes dynamically. In our model the unit intervals are added or removed one at the time, and have to be colored immediately, so that no two overlapping intervals share the same color. After each update only a limited number of intervals are allowed to be recolored. The limit on the number of recolorings per update is called the recourse budget. In this paper we show, that if the graph remains k -colorable at all times, the updates consist of insertions only, and the final instance consists of n intervals, then we can achieve an amortized recourse budget of $\mathcal{O}(k^7 \log n)$ while maintaining a proper coloring with k colors. This is an exponential improvement over the result in [10] in terms of both k and n . We complement this result by showing the lower bound of $\Omega(n)$ on the amortized recourse budget in the fully dynamic setting. Our incremental algorithm can be efficiently implemented.

As an additional application of our techniques we include a new combinatorial result on coloring unit circular arc graphs. Let L be the maximum number of arcs intersecting in one point for some set of unit circular arcs \mathcal{A} . We show that if there is a set \mathcal{A}' of non-intersecting unit arcs of size $L^2 - 1$ such that $\mathcal{A} \cup \mathcal{A}'$ does not contain $L + 1$ arcs intersecting in one point, then it is possible to color \mathcal{A} with L colors. This complements the work on circular arc coloring [4, 30, 31], which specifies sufficient conditions needed to color \mathcal{A} with $L + 1$ colors or more.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases dynamic algorithms, unit interval graphs, coloring, recourse budget, parametrized dynamic algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.25

Related Version *Full Version*: <https://arxiv.org/abs/2202.08006>

Funding National Science Centre, Poland, Grant 2017/26/D/ST6/00264.

Acknowledgements The authors want to thank an anonymous reviewer for very useful comments.

1 Introduction

In this paper we study dynamic algorithms for the graph coloring problem. The setting we focus on is as follows. We are given a graph G , which is modified over time by vertex insertions or vertex deletions, where vertices are inserted (or deleted) together with all the adjacent edges connecting them to the vertices that are already present in G . For a positive integer k , a *proper k -coloring* of a graph is an assignment of colors in $\{1, \dots, k\}$ to the vertices of the graph in such a way that no two adjacent vertices share a color. We say that a graph admitting such an assignment is k -colorable. In the dynamic setting, the ultimate goal is to design an algorithm, that (for some given $l \geq k$) efficiently maintains the proper l -coloring on a dynamically changing k -colorable graph G .

¹ corresponding author



Dynamic graph coloring is a fundamental problem in computer science and as such it has received a lot of attention in the literature. Chronologically, dynamic graph coloring was first considered in a more restricted online setting, where the updates consist of vertex insertions only, and one is not allowed to change the colors of the previously added vertices. Many pessimistic lower bounds for online graph coloring have been revealed. In particular, for general k -colorable graphs, one cannot color the graph online with less than $l = \frac{n}{\log^2 n} k$ colors [18], where n is the number of vertices added to G . This lower bound holds even for randomized algorithms. Even for trees, which are 2-colorable, one needs $l = \Omega(\log n)$ colors [3] in the online model. The situation improves if we consider k -colorable interval graphs, for which $l = 3k - 2$ colors are necessary and sufficient in the online model [20]. The case of unit interval graphs has also been extensively studied, revealing that $2k - 1$ colors suffice and $3k/2$ colors are necessary to color a k -colorable unit interval graph online [6, 15].

To go beyond the pessimistic lower bounds imposed by the online model, several settings were proposed where an algorithm is given more power. The one that received a lot of attention is the limited recourse budget framework. In this setting the algorithm is allowed to change a number of past decisions, but there is a limit on the number of such changes, referred to as the recourse budget. This model has been well established and successfully applied to a variety of optimization problems, often implying efficient dynamic algorithms [5, 7, 8, 9, 19, 23, 21, 2, 29, 10]. For the coloring problem that we study, the recourse budget is the number of vertices that change their color after an addition or removal of a vertex. Barba et al. apply the recourse budget model to coloring general graphs [2]. They devise two complementary algorithms. For any $d > 0$, the first (resp. second) algorithm maintains a $k(d+1)$ -coloring (resp. $k(d+1)n^{1/d}$ -coloring) of a k -colorable graph and recolors at most $(d+1)n^{1/d}$ (resp. d) vertices per update, which is either addition or removal of a vertex. While the second trade-off was improved in [29], the authors in [2] show that the first trade-off is tight, and the bad example is a forest. Thus, if one insists on using few colors, one has to incur a polynomial in n recourse budget on every class of graphs that contains forests. This pushed the researchers to apply the limited recourse budget model to coloring interval and unit interval graphs.

For unit interval graphs a very positive result has been obtained. The recoloring budget of $\mathcal{O}(k^2)$ (worst case) is sufficient for maintaining a $(k+1)$ -coloring of a k -colorable graph [10]. It is left open what budget is needed for maintaining an optimal k -coloring for unit interval graphs, even if we only allow vertex insertions. The lower bound given in [10] is $\Omega(\log n)$ (even when updates are only insertions), while the upper bound (which only works for vertex insertions) is $\mathcal{O}(k!\sqrt{n})$ (the same as for general interval graphs). Such a tremendous gap for such elementary graph class calls for further investigation. The main result of this paper is that we close this gap up to the factors polynomial in k . To be more precise, we show that an amortized recourse budget of $\mathcal{O}(k^7 \log n)$ is sufficient to maintain k -coloring under vertex insertions. This is an exponential improvement over [10] in terms of both n and k . It is fairly easy to see that our algorithm can be efficiently implemented. We complement this result by showing that in the fully dynamic setting one must spend an amortized recourse budget of $\Omega(n)$ per update.

It is worth emphasizing, that our results show a fine line between $(k+1)$ -coloring and optimal k -coloring of unit interval graphs in the limited recourse budget model. While $(k+1)$ -coloring admits an algorithm with the worst case recourse budget of $\mathcal{O}(k^2)$ in the fully dynamic setting, we cannot hope (in this setting) for any reasonable recourse budget for optimal k -coloring, even if we allow amortization. If we restrict to only adding intervals, we get the amortized recourse budget of $\mathcal{O}(k^7 \log n)$ and $\Omega(\log n)$ is the lower bound. The

incremental setting is interesting by itself, as it generalizes the online model (where the recourse budget is zero and only insertions are allowed). In other words, we show that if in the online model we allow a modest number of $\mathcal{O}(k^7 \log n)$ recolorings per update, we can get down from $3k/2$ to the optimum number k of colors. Our result is not only motivated by the online model, but also fits nicely in the recent line of research related to parameterized dynamic algorithms [1, 14, 12], with k being the parameter.

The techniques we develop to obtain our main result seem applicable to a wider range of problems. In particular, we apply one of our techniques to the problem of coloring unit circular arc graphs. We obtain a combinatorial result that nicely fits into the related line of research. Imagine that L is the maximum number of arcs intersecting in one point for some set of unit circular arcs \mathcal{A} . We show that if there is a set \mathcal{A}' of non-intersecting unit arcs of size $L^2 - 1$ such that $\mathcal{A} \cup \mathcal{A}'$ does not contain $L + 1$ arcs intersecting in one point, then it is possible to color \mathcal{A} with L colors. This complements the work on circular arc coloring [4, 30, 31], which specifies sufficient conditions needed to color \mathcal{A} with $L + 1$ colors or more.

The remainder of the paper is organized as follows. In Section 2 we provide basic definitions related to interval graphs and unit interval graphs, together with some elementary algorithms that solve the coloring problem for static instances of those graphs. In Section 3 we provide an overview of our results and techniques. Due to space limitations, we only sketch the proofs in Section 3, skipping most of technical details. The full proofs can be found in the full version of the paper [11]. We conclude with Section 4 in which we discuss some properties and extensions of our main result. In particular, we discuss the implementation of our incremental algorithm and its actual running time (which is amortized $\mathcal{O}(k^7 \log n)$ per interval insertion). We also discuss extending our incremental algorithm to changing values of k , and finally we briefly discuss why our main result does not easily extend to general interval graphs. We end Section 4 with the problems that we leave open.

2 Preliminaries

We consider in this paper closed-open intervals $I = [x, y)$ for some $x, y \in \mathbb{R}, x < y$. Note that this causes no loss in generality, as closed-open intervals induce the same class of graphs as open-closed, closed-closed and open-open intervals (see [13, 26]). For an interval I we define operators $x(I) \stackrel{\text{def}}{=} x$ and $y(I) \stackrel{\text{def}}{=} y$ for accessing the begin and the end coordinate. A multiset of intervals can be interpreted as a graph: the intervals are interpreted as vertices, which are adjacent if and only if the corresponding intervals intersect. Graphs obtained in this way are called interval graphs. The coloring related definitions stated in the first paragraph of Section 1 directly translate to multisets of intervals interpreted as graphs. For a multiset of intervals \mathcal{I} , the function $c : \mathcal{I} \mapsto \{1, \dots, k\}$ is a proper k -coloring if for any $I, J \in \mathcal{I}$ it holds that $c(I) \neq c(J)$ if I and J intersect. The chromatic number $\chi(\mathcal{I})$ is the minimum number k for which \mathcal{I} admits a proper k -coloring. Similarly we can adapt the definition of a clique: a multiset of intervals $\mathcal{J} = \{J_1, J_2, \dots, J_m\}$ is a clique if and only if $\bigcap \mathcal{J} \stackrel{\text{def}}{=} J_1 \cap \dots \cap J_m \neq \emptyset$, or equivalently $\max_{i=1}^m x(J_i) < \min_{i=1}^m y(J_i)$. In such case it holds that $\bigcap \mathcal{J} = [\max_{i=1}^m x(J_i), \min_{i=1}^m y(J_i))$. We refer to this intersection as span of \mathcal{J} , and denote it as $\text{span}(\mathcal{J}) \stackrel{\text{def}}{=} \bigcap \mathcal{J}$. To emphasize the size of \mathcal{J} we often refer to it as an m -clique. The clique number $\omega(\mathcal{I})$ of a multiset of intervals \mathcal{I} is the maximum number m such that \mathcal{I} contains an m -clique. It is well known that interval graphs are perfect, that is:

► **Lemma 1** (Golumbic [17]). *Let \mathcal{I} be a multiset of intervals. Then $\omega(\mathcal{I}) = \chi(\mathcal{I})$.*

We introduce two orders \sqsubset and $<$ on a multiset of intervals \mathcal{I} . For any $I, J \in \mathcal{I}$ we let $I \sqsubset J$ if $x(I) < x(J)$. If $I = J$, we solve the tie arbitrarily. Hence, \sqsubset is a linear order on \mathcal{I} . We say that $I < J$ if and only if $y(I) \leq x(J)$. Hence, $<$ is a linear order only on independent sets of intervals (i.e., multisets of intervals that are pairwise non-intersecting). Orders \sqsubset and $<$ extend to multisets of intervals in a natural manner. For two multisets of intervals \mathcal{J} and \mathcal{K} , we say that $\mathcal{J} \sqsubset \mathcal{K}$ (respectively $\mathcal{J} < \mathcal{K}$) if for all $J \in \mathcal{J}, K \in \mathcal{K}$ it holds that $J \sqsubset K$ (respectively $J < K$). For a multiset of intervals \mathcal{I} by $\mathcal{I} = \{I_1 \sqsubset \dots \sqsubset I_m\}$ we denote that $\mathcal{I} = \{I_1, \dots, I_m\}$ and $I_1 \sqsubset \dots \sqsubset I_m$. For an ordered multiset of intervals $\mathcal{I} = \{I_1 \sqsubset \dots \sqsubset I_m\}$ we now define a prefix, a suffix and an infix of \mathcal{I} . For $I_i, I_l \in \mathcal{I}, l > i$ we set $\text{prefix}_{\mathcal{I}}^{\sqsubset}(I_i) = \{I_1, \dots, I_i\}$, $\text{suffix}_{\mathcal{I}}^{\sqsubset}(I_i) = \{I_i, \dots, I_m\}$ and $\text{infix}_{\mathcal{I}}^{\sqsubset}(I_i, I_l) = \{I_i, \dots, I_l\}$.

We now move on to some basic observations that hold for multisets of unit intervals. An interval I is unit if and only if $y(I) = x(I) + 1$.

► **Observation 2.** *Let $\mathcal{I} = \{I_1 \sqsubset \dots \sqsubset I_m\}$ be a multiset of unit intervals. If $\omega(\mathcal{I}) < m$, then the extremal intervals are disjoint, i.e., $I_1 < I_m$.*

Proof. Suppose contrary that $I_1 \cap I_m \neq \emptyset$, i.e., $x(I_m) < y(I_1) = x(I_1) + 1$. Then $\max_{i=1}^m x(I_i) < \min_{i=1}^m y(I_i)$ and as a consequence $\bigcap \mathcal{I} \neq \emptyset$ contradicting $\omega(\mathcal{I}) < m$. ◀

For interval graphs there is a simple greedy algorithm, that can be applied to complete a coloring given on the prefix of the representation ordered by \sqsubset [24]. In this paper we need a more specific but equally simple algorithm which is restricted to unit interval graphs. The same idea was used for instance to color proper circular arc graphs [25] or to schedule round-robin tournaments [27]. Since we use it in a different context, we introduce it here from scratch and we refer to it as the MODULO COLOR COMPLETION algorithm. Informally, given some coloring on the prefix of a k -colorable instance, this algorithm looks at the first k consecutive uncolored intervals (in \sqsubset order), and copies the coloring given on the last k intervals that are colored (respecting the \sqsubset order). This proceeds until all intervals are colored. This simple procedure works given that the coloring that is being copied consists of all colors from 1 to k . We describe the MODULO COLOR COMPLETION algorithm more formally in the following observation, which also proves the correctness.

► **Observation 3.** *Let $\mathcal{I} = \{I_1 \sqsubset \dots \sqsubset I_m\}$ be a multiset of unit intervals such that $\omega(\mathcal{I}) \leq k$. Let $k \leq l < m$ and let $c : \text{prefix}_{\mathcal{I}}^{\sqsubset}(I_l) \mapsto [k]$ be a proper k -coloring for $\text{prefix}_{\mathcal{I}}^{\sqsubset}(I_l)$ such that c is a bijection on $\text{infix}_{\mathcal{I}}^{\sqsubset}(I_{l-k+1}, I_l)$. Let $c' : \mathcal{I} \mapsto [k]$ (referred to as MODULO COLOR COMPLETION) be defined as follows: $c'(I_i) \stackrel{\text{def}}{=} c(I_i)$ for $i \in [l]$ and $c'(I_{l+i}) \stackrel{\text{def}}{=} c(I_{l-k+(i \bmod k)})$ for $i \in [m-l]$. Then c' is a proper k -coloring on \mathcal{I} .*

Proof. It is clear that c assigns only colors in $[k]$, it remains to prove that it is also a proper coloring. Let $I_i, I_j \in \mathcal{I}$, where $i < j$ and $j > l$. If $j < i + k$, then by definition $c(I_i) \neq c(I_j)$. Otherwise, $|\{I_i, I_{i+1}, \dots, I_j\}| \geq k + 1$, so by Observation 2, $I_i \cap I_j = \emptyset$. ◀

The following observation will be useful to bring the prefix coloring to the state when we can use the MODULO COLOR COMPLETION algorithm, i.e., the coloring on the prefix ends with the bijection.

► **Observation 4.** *Let $\mathcal{I} = \{I_1 \sqsubset \dots \sqsubset I_{2k}\}$ be a multiset of unit intervals such that $\omega(\mathcal{I}) \leq k$ and let $c' : \text{prefix}_{\mathcal{I}}^{\sqsubset}(I_k) \mapsto [k]$ be a proper k -coloring on $\text{prefix}_{\mathcal{I}}^{\sqsubset}(I_k)$. Then there is a proper k -coloring $c : \mathcal{I} \mapsto [k]$ such that $c(I_i) = c'(I_i)$ for $i \in [k]$ and c is a bijection on $\text{suffix}_{\mathcal{I}}^{\sqsubset}(I_{k+1})$.*

Proof. To construct c , we first set $c(I_i) = c'(I_i)$ for all $i \in [k]$. Let $\mathcal{J} = \{I_l, I_{l+1}, \dots, I_k\}$ be the intervals of $\text{prefix}_{\mathcal{I}}^{\sqsubset}(I_k)$ that intersect I_k . Thus, \mathcal{J} is a $(k-l+1)$ -clique. We first assign the $l-1$ colors not used by \mathcal{J} to the first $l-1$ intervals of $\text{suffix}_{\mathcal{I}}^{\sqsubset}(I_{k+1})$. That is we assign

colors $[k] \setminus c(\mathcal{J})$ to intervals $\text{infix}_{\mathcal{I}}^{\square}(I_{k+1}, I_{k+l-1})$ in an arbitrary order. We get a proper coloring since the intervals of $\text{infix}_{\mathcal{I}}^{\square}(I_{k+1}, I_{k+l-1})$ do not intersect intervals of $\text{prefix}_{\mathcal{I}}^{\square}(I_k)$ other than \mathcal{J} . Finally, we set $c(I_i) = c(I_{i-k})$ for $i \in \{k+l, \dots, 2k\}$. By Observation 2 intervals I_i and I_{i-k} do not intersect. This completes the proof. \blacktriangleleft

Observe that Observation 3 and Observation 2 give an alternative to [24] algorithm that completes the prefix coloring on unit interval graphs. We refer to this algorithm as GREEDY COLOR COMPLETION (even though we can use the algorithm of [24] instead).

3 Overview of our results and techniques

Our final result is based on new techniques that might be of independent interest. This section outlines our techniques and gives an overview on how they are combined into obtaining the final result. Our techniques are encapsulated in Section 3.1 and Section 3.2 as completely independent results, as we see the potential of applying them to a wider range of problems. In Section 3.1 we introduce our new technique of *color sorting*, which allows to solve the UNIT PRECOLORING EXTENSION problem efficiently under a specific condition that might naturally appear in a number of applications. In Section 3.2 we introduce the FROGS game technique, which is a natural generalization of the folklore technique applied for merging sets in Find-Union like algorithms. We expect the FROGS game technique to be also applicable to a wider range of problems. Our main application of the color sorting technique and the FROGS game technique is presented in Section 3.3, where we introduce the INCREMENTAL UNIT INTERVAL RECOLORING problem and sketch the solution to this problem.

3.1 Color sorting applied to the Unit Precoloring Extension problem

Our first contribution is the color sorting technique, that we apply to the UNIT PRECOLORING EXTENSION problem. In this problem we are given a k -colorable multiset of unit intervals $\mathcal{I} = \{I_1 \sqsubset \dots \sqsubset I_m\}$, $m > 2k$. We are also given a proper k -coloring c' on $\text{prefix}_{\mathcal{I}}^{\square}(I_k)$ (the first k intervals) and a proper k -coloring c'' on $\text{suffix}_{\mathcal{I}}^{\square}(I_{m-k+1})$ (the last k intervals). The problem is to extend c' and c'' to a proper l -coloring of \mathcal{I} minimizing l .

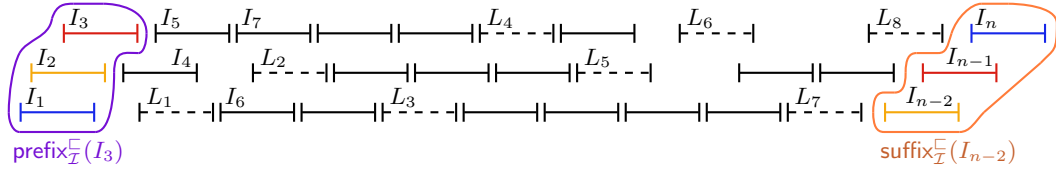
The UNIT PRECOLORING EXTENSION problem was proven NP-hard [22], moreover, it is even NP-hard to decide whether one can extend the coloring using $l = k$ colors. In our application, however, we are only interested in the instances when this is possible, i.e., the UNIT PRECOLORING EXTENSION ceases to be NP-hard. We specify a condition on \mathcal{I} under which one can, using the color sorting technique, extend the precoloring to a proper k -coloring. Our condition essentially requires that there is some slack space between the colored prefix and the colored suffix, which allows color sorting. By the slack we mean that we can fit between the prefix and the suffix additional $k^2 - 1$ mutually disjoint unit intervals without increasing the chromatic number of \mathcal{I} . This slack is used to gradually sort the color permutation given on the prefix to finally obtain the color permutation given on the suffix, in an insertion sort fashion. The precise result is stated in the following lemma.

► **Lemma 5** (Precoloring Extension Lemma). *Let k be an integer and let $\mathcal{I} = \{I_1 \sqsubset \dots \sqsubset I_m\}$ be a k -colorable multiset of $m \geq 2k$ unit intervals. Let $c' : \text{prefix}_{\mathcal{I}}^{\square}(I_k) \mapsto [k]$ and $c'' : \text{suffix}_{\mathcal{I}}^{\square}(I_{m-k+1}) \mapsto [k]$ be bijections. Let there exist a set of $k^2 - 1$ pairwise non-intersecting unit intervals \mathcal{L} , such that $\text{prefix}_{\mathcal{I}}^{\square}(I_k) \sqsubset \mathcal{L} \sqsubset \text{suffix}_{\mathcal{I}}^{\square}(I_{m-k+1})$ and $\mathcal{I} \cup \mathcal{L}$ is k -colorable. Then there is a proper k -coloring $c : \mathcal{I} \mapsto [k]$ such that c extends both c' and c'' , i.e., $c(I_i) = c'(I_i)$ for $i \in [k]$ and $c(I_i) = c''(I_i)$ for $i \in \{m - k + 1, \dots, m\}$.*

The full proof of Lemma 5 can be found in the full version of the paper [11], Appendix A.

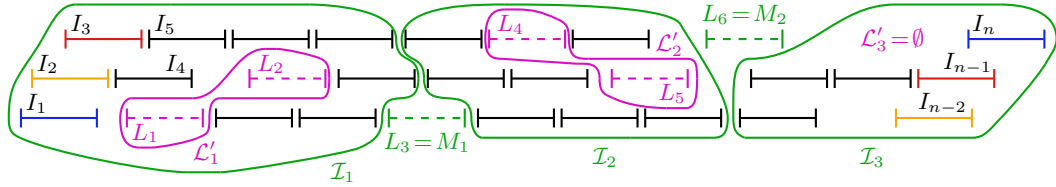
Sketch of proof of Lemma 5. The proof introduces the color sorting technique, which uses the slack intervals of \mathcal{L} to gradually transform the coloring c' into the coloring c'' , in an insertion sort like fashion. In order to accomplish that, we modify the instance (by inserting dummy intervals from \mathcal{L}) to allow the color sorting technique to work. We describe this process first.

We start by assuming that the multiset of unit intervals \mathcal{I} is connected as a graph (otherwise Lemma 5 follows from the MODULO COLOR COMPLETION algorithm, see Observation 3). An example instance for which our assumptions hold is pictured in Figure 1.



■ **Figure 1** An example illustrating the assumptions of Lemma 5 for $k = 3$.

We now partition $\mathcal{L} = \{L_1 < \dots < L_{k^2-1}\}$ into $2k - 1$ sets of unit intervals as follows: $\mathcal{L} = \mathcal{L}_1 \cup \{M_1\} \cup \mathcal{L}_2 \cup \dots \cup \{M_{k-1}\} \cup \mathcal{L}_k$, where $\mathcal{L}_i = \{L_{(i-1)k+1}, \dots, L_{ik-1}\}$ for $i \in [k]$ and $M_i = L_{ik}$ for $i \in [k-1]$. Observe that $|\mathcal{L}_i| = k - 1$. Thus, the intervals of the partition are ordered as follows: $\text{prefix}_{\mathcal{I}}^{\square}(I_k) \sqsubset \mathcal{L}_1 < \{M_1\} < \mathcal{L}_2 < \dots < \{M_{k-1}\} < \mathcal{L}_k \sqsubset \text{suffix}_{\mathcal{I}}^{\square}(I_{n-k+1})$. Next we partition \mathcal{I} into k parts: $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2 \cup \dots \cup \mathcal{I}_k$, in the way that the following holds: $\text{prefix}_{\mathcal{I}}^{\square}(I_k) \subseteq \mathcal{I}_1 \sqsubset \{M_1\} \sqsubset \mathcal{I}_2 \sqsubset \dots \sqsubset \{M_{k-1}\} \sqsubset \mathcal{I}_k \supseteq \text{suffix}_{\mathcal{I}}^{\square}(I_{n-k+1})$. This partition is pictured in Figure 2. We now enlarge \mathcal{I} by adding to each \mathcal{I}_i some dummy intervals. More



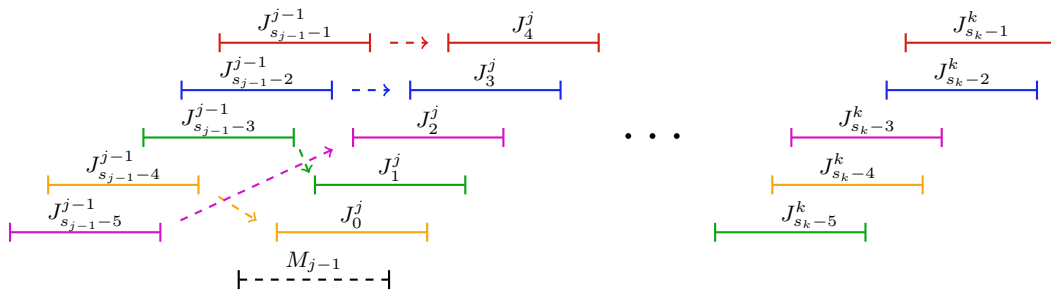
■ **Figure 2** An example illustrating partitioning the intervals, $k = 3$.

precisely, to each $\mathcal{I}_i \subseteq \mathcal{I}$ we add a subset $\mathcal{L}'_i \subseteq \mathcal{L}_i$ as to make the number of intervals in the extension $\mathcal{J}_i \stackrel{\text{def}}{=} \mathcal{I}_i \cup \mathcal{L}'_i$ a multiple of k (see Figure 2). As a consequence, $\mathcal{J} \stackrel{\text{def}}{=} \bigcup_{i=1}^k \mathcal{J}_i$ and $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_k$ satisfy the following (note that since \mathcal{I} is connected as a graph, each \mathcal{I}_i is nonempty and thus each \mathcal{J}_i is also nonempty):

1. $\omega(\mathcal{J} \cup \{M_1, \dots, M_{k-1}\}) \leq k$,
2. for each $i \in [k]$ we have $|\mathcal{J}_i| = kp_i$ for some $p_i \in \mathbb{N} \setminus \{0\}$,
3. $\text{prefix}_{\mathcal{I}}^{\square}(I_k) \subseteq \mathcal{J}_1 \sqsubset \{M_1\} \sqsubset \mathcal{J}_2 \sqsubset \dots \sqsubset \{M_{k-1}\} \sqsubset \mathcal{J}_k \supseteq \text{suffix}_{\mathcal{I}}^{\square}(I_{n-k+1})$.

Now, rather than the proper coloring for \mathcal{I} , we construct the proper coloring c for \mathcal{J} (which is obviously also proper for \mathcal{I}). The construction of c starts by copying colors of c' , so that c coincides with c' on $\text{prefix}_{\mathcal{I}}^{\square}(I_k)$. Let us now consider a block $\mathcal{J}_{j-1} = \{J_0^{j-1} \sqsubset \dots \sqsubset J_{s_{j-1}-1}^{j-1}\}$, where $(j-1) \in [k]$, $s_{j-1} = |\mathcal{J}_{j-1}|$. Suppose that c is already defined on $\text{prefix}_{\mathcal{J}_{j-1}}^{\square}(J_{k-1}^{j-1})$ (the first k intervals of \mathcal{J}_{j-1}). Note that coloring c on $\text{prefix}_{\mathcal{J}_{j-1}}^{\square}(J_{k-1}^{j-1})$ defines a permutation of colors. We use the MODULO COLOR COMPLETION algorithm to copy this permutation to $\text{suffix}_{\mathcal{J}_{j-1}}^{\square}(J_{s_{j-1}-k}^{j-1})$ (the last k intervals of \mathcal{J}_{j-1}). This works because $k | s_{j-1}$. Suppose

that this permutation already coincides with the permutation given by c'' on the last $j - 2$ positions. We now use the slack given by M_{j-1} to define c on $\text{prefix}_{\mathcal{J}_j}^{\square}(J_{k-1}^j)$ in such a way that the corresponding permutation agrees with the permutation given by c'' on the last $j - 1$ positions. To be more precise, we apply one step of the insertion sort on the permutation to bring one more color to the appropriate position, as shown in Figure 3. This is possible because of the slack given by M_{j-1} . The intuition is that if we apply MODULO COLOR COMPLETION on $\text{prefix}_{\mathcal{J}_j}^{\square}(J_{k-1}^j)$, then we obtain a proper k -coloring on $\text{prefix}_{\mathcal{J}_j}^{\square}(J_{k-1}^j)$ that precisely copies the color permutation from $\text{suffix}_{\mathcal{J}_{j-1}}^{\square}(J_{s_{j-1}-k}^{j-1})$. On the other hand, if we apply MODULO COLOR COMPLETION to $\{M_{j-1}\} \cup \text{prefix}_{\mathcal{J}_j}^{\square}(J_{k-1}^j)$, then on $\text{prefix}_{\mathcal{J}_j}^{\square}(J_{k-1}^j)$ we get the same permutation with all colors shifted down by one, and this is also a proper coloring. The insertion sort step that we apply (see Figure 3) gives a permutation that alternates at most twice between the same permutation and the shifted permutation. Also, the insertion sort step moves one special color (purple in Figure 3) further away than required by the MODULO COLOR COMPLETION algorithm. Hence the obtained coloring is proper. \blacktriangleleft



■ **Figure 3** An insertion sort step, where coloring c is constructed on $\text{prefix}_{\mathcal{J}_j}^{\square}(J_{k-1}^j)$ for $j = 4$, $k = 5$.

3.2 The Frogs game

Our second contribution is a technique that generalizes a folklore trick typically used in the analysis of Union-Find data structure. The SET UNION problem, where the Union-Find data structure finds its application, can be thought of as a game. In this game we are initially given n pairwise disjoint sets of size δ , and the adversary keeps merging consecutive pairs of sets until there is only one set left. Each time the adversary merges a pair of sets, we incur the cost equal to the size of the smaller set among the ones being merged (as if we move all elements of the smaller set to the larger set, one by one). It is clear that the maximum total cost the adversary can generate is $\delta n \log n$, as each of δn elements can contribute to the total cost at most $\log n$ times.

The generalization we propose is that the cost we incur with each merging is the sum of the sizes of κ consecutive sets rather than just one. We sum κ consecutive sizes either to the left or to the right of the merged pair, whatever turns out cheaper. We show that the total cost an adversary can generate here is $\mathcal{O}(\delta \kappa n \log n)$. We refer to this generalization as the FROGS game, and we now introduce its formal definition.

► **Definition 6.** We define an instance of a FROGS game as a tuple $\mathcal{F} = (N, \kappa, \delta, J)$. There, N, κ , and δ are integers, referred to as the size of the game, the jump number, and the initial rank value respectively. It also holds that $\kappa \leq N$. Moreover J is a sequence of $N - 1$ integers $J = (j_1, j_2, \dots, j_{N-1})$, where $1 \leq j_\tau \leq N - \tau$ and J is referred to as the jump sequence.

We now define the cost of the FROGS game.

► **Definition 7.** Let $\mathcal{F} = (N, \kappa, \delta, J)$ be an instance of the FROGS game, where $J = (j_1, j_2, \dots, j_{N-1})$. Let $R_1 = (\delta, \dots, \delta)$ be a sequence of length N (R_1 is called the initial rank sequence). Let $R_{\tau+1} \stackrel{\text{def}}{=} (r_{\tau,1}, \dots, r_{\tau,j_\tau-1}, r_{\tau,j_\tau} + r_{\tau,j_\tau+1}, r_{\tau,j_\tau+2}, \dots, r_{\tau,N-\tau+1})$ (rank sequence $R_{\tau+1}$ in time $\tau + 1$ is obtained by adding two neighboring ranks in R_τ placed at positions j_τ and $j_\tau + 1$). The FROGS cost incurred in time τ is $\varsigma_\tau \stackrel{\text{def}}{=} \min(r_{\tau,j_\tau-\kappa+1} + \dots + r_{\tau,j_\tau}, r_{\tau,j_\tau+1} + \dots + r_{\tau,j_\tau+\kappa})$, where for $i < 1$ and $i > N - \tau + 1$ we set $r_{\tau,i} \stackrel{\text{def}}{=} \delta$. The total cost of the FROGS game \mathcal{F} is $\varsigma(\mathcal{F}) \stackrel{\text{def}}{=} \varsigma_1 + \dots + \varsigma_{N-1}$.

We bound the cost of any FROGS game \mathcal{F} using the following theorem.

► **Theorem 8 (Frogs theorem).** For an instance $\mathcal{F} = (N, \kappa, \delta, J)$ of the FROGS game we have

$$\varsigma(\mathcal{F}) \leq \delta(2\kappa - 1)(N + 2\kappa - 2) \log_2 \frac{N + 2\kappa - 2}{2\kappa - 1}.$$

The FROGS theorem is proved in the full version of the paper [11] in Appendix B. The technique given by FROGS theorem might be of independent interest. It seems applicable as a building block to more problems than the one studied further in this paper.

3.3 Incremental Unit Interval Recoloring problem

The main result of this paper is an algorithm for the INCREMENTAL UNIT INTERVAL RECOLORING problem. We are given a parameter k and a sequence of n unit intervals: I_1, I_2, \dots, I_n such that $\{I_1, \dots, I_n\}$ is k -colorable. This sequence defines $n + 1$ multisets of unit intervals: $\mathcal{I}_0 = \emptyset$ and $\mathcal{I}_j \stackrel{\text{def}}{=} \{I_1, \dots, I_j\}$ for $j > 0$. Instance \mathcal{I}_j differs from \mathcal{I}_{j-1} by one interval I_j . The goal is to maintain a proper k -coloring on the dynamic instance. To be more precise, after the interval I_j is presented, the algorithm needs to compute a proper k -coloring c_j for \mathcal{I}_j . Our objective is to minimize the recourse budget, which is the number of intervals with different colors in c_j and c_{j-1} . We obtain the following result.

► **Theorem 9.** There is an algorithm for the INCREMENTAL UNIT INTERVAL RECOLORING problem with a total recourse budget of $\mathcal{O}(k^7 n \log n)$.

Theorem 9 is formally proved in the full version of the paper [11], Appendix C. Here we outline how the Precoloring Extension lemma (Lemma 5) and the Frogs theorem (Theorem 8) are combined to obtain the main result, skipping the technical details that are deferred to the full proof.

Sketch of the proof of Theorem 9. Let us imagine that a new interval I_j is presented and we need to provide the coloring c_j for \mathcal{I}_j . Let us order \mathcal{I}_j according to the \sqsubset order. Let I be the direct predecessor (in \sqsubset order) of I_j in \mathcal{I}_j . Let interval $J^{(R)}$ be the interval of \mathcal{I}_j succeeding I (in \sqsubset order) such that between I and $J^{(R)}$ there is room to insert the set \mathcal{L} of $k^2 - 1$ mutually disjoint unit intervals, as required by the Precoloring Extension lemma (Lemma 5). Let $J^{(L)}$ be an analogous interval preceding I . For simplicity let us assume that both $J^{(L)}$ and $J^{(R)}$ exist. Based on the Precoloring Extension lemma (Lemma 5), it is (almost) sufficient to recolor the infix of \mathcal{I}_j between I and $J^{(R)}$, or the infix of \mathcal{I}_j between $J^{(L)}$ and I , and it is up to the algorithm which of the two infixes it chooses to recolor. Our recoloring algorithm basically chooses the infix that is smaller in size and recolors it (although there are small technical details that make the algorithm a bit more complicated in the end).

The whole weight of the proof lies now in the analysis, which shows that the size of the smaller infix is $\mathcal{O}(k^7 \log n)$ in the amortized sense. To prove that we use the Frogs theorem (Theorem 8), so we need to define the appropriate size N , jump number κ , initial rank value δ , and the jump sequence J , and all these numbers must strongly relate to what the recoloring algorithm does. We sketch here the main idea of how we do it. We look at all k -cliques of the final instance $\mathcal{I}_{n'}$, $n' \in \mathcal{O}(kn)$ (the final instance is not \mathcal{I}_n because, for a technical reason mentioned further, we may be forced to add some dummy intervals after the whole instance \mathcal{I}_n was presented). Every k -clique is a set of k overlapping intervals, whose intersection is some (not necessarily unit) interval. So we represent the k -cliques of the final instance $\mathcal{I}_{n'}$ as a set $\mathcal{S}_{n'}$ of intervals, which are mutually disjoint, and we order them by $<$ order. This is shown in Figure 4, where at the bottom the intervals of $\mathcal{I}_{n'}$ are drawn (either solid black or dashed blue), and above them the corresponding cliques of $\mathcal{S}_{n'}$ are pictured (also either solid black or dashed blue).

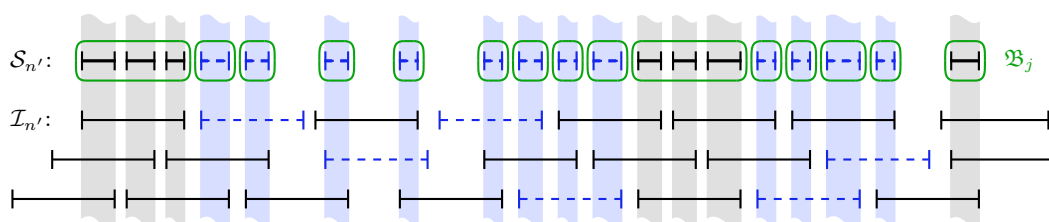


Figure 4 Example span partition : — – intervals of \mathcal{I}_j , - - - – intervals of $\mathcal{I}_{n'} \setminus \mathcal{I}_j$ (future intervals), — – spans of \mathcal{S}_j , - - - – spans of $\mathcal{S}_{n'} \setminus \mathcal{S}_j$ (future spans), ○ – blocks of partition \mathfrak{B}_j , $k = 3$.

We observe that $m \stackrel{\text{def}}{=} |\mathcal{S}_{n'}| = \mathcal{O}(kn)$. We assume that $\mathcal{S}_{n'}$ is tightly packed, i.e., two consecutive intervals in $\mathcal{S}_{n'}$ are at distance less than one (for this assumption to hold we add dummy intervals to \mathcal{I}_n and obtain $\mathcal{I}_{n'}$ -whenever two consecutive cliques are at distance at least one, we insert a dummy interval between them without increasing the chromatic number). The intuition now is that at the beginning each clique in $\mathcal{S}_{n'}$ is empty, but successively the cliques in $\mathcal{S}_{n'}$ are filled with the intervals, until each of them is filled up to the maximum level k . A new interval I_j adds 1 to the level of all the cliques it intersects. Let us denote by $\text{lvl}_j(S)$ the level of clique $S \in \mathcal{S}_{n'}$ in step j . In Figure 4, the intervals of \mathcal{I}_j are marked solid black, while the future intervals (not presented until step j) are marked dashed blue. Consequently, the fully filled cliques of $\mathcal{S}_{n'}$ are marked solid black, while the cliques that are not filled to the maximum level in step j are marked dashed blue.

In each step j , we partition the cliques $S \in \mathcal{S}_{n'}$ into blocks depending on their level $\text{lvl}_j(S)$. We refer to the corresponding partition as \mathfrak{B}_j . The rule is that the consecutive cliques who are entirely filled (meaning that $\text{lvl}_j(S) = k$) belong to the same block of the partition \mathfrak{B}_j , while the cliques that are not filled are placed in a separate one-element block. An example partition \mathfrak{B}_j is pictured in Figure 4. As a result, the blocks of \mathfrak{B}_j are merged over the time, but never split. The grey blocks of \mathfrak{B}_j consisting of entirely filled cliques are called passive, while the remaining blue blocks (the one-element blocks containing the cliques that are not filled) are called active.

Now for each insertion step j there is a corresponding rank sequence R_τ in the FROGS game. Each block $\mathcal{B} \in \mathfrak{B}_j$ is assigned at least one and at most $(k + 1)$ consecutive ranks $r_{\tau,i}^{(\mathcal{B})}$ in R_τ , in the way that every rank $r_{\tau,i}^{(\mathcal{B})}$ assigned to \mathcal{B} bounds the sum of the levels in \mathcal{B} , i.e., $r_{\tau,i}^{(\mathcal{B})} \geq \sum_{S \in \mathcal{B}} \text{lvl}_j(S)$. Each active block \mathcal{B} is assigned at least $k - \text{lvl}_j(S) + 1$ consecutive ranks $r_{\tau,i}^{(\mathcal{B})}$, while each passive block is assigned precisely one rank.

When the interval I_j arrives, some cliques $S \in \mathcal{S}_n$ increase their level. Such a clique S that increases its level is necessarily in an active (singleton) block $\mathcal{B} = \{S\}$, as I_j certifies that S is not completely filled up yet. To account for the insertion of I_j , we merge any two consecutive ranks assigned to $\mathcal{B} = \{S\}$ (there is at least $k - \text{lvl}_j(S) + 1 \geq 2$ ranks available). We observe that the infix recolored by the algorithm is entirely contained within $\mathcal{O}(k^3)$ consecutive blocks to the left or to the right of \mathcal{B} , and in what follows we argue why this holds. First of all, the recolored infix can intersect at most $\mathcal{O}(k^3)$ active blocks: each active block certifies that a future unit interval fits in, but k active blocks may be witnesses for the same future unit interval that fits in. Note that the infix recolored by the algorithm contains at most $(k^2 - 1)$ pairwise disjoint future intervals. Second of all, the recolored infix intersects no more passive blocks than active blocks, since each passive block has a neighboring active block.

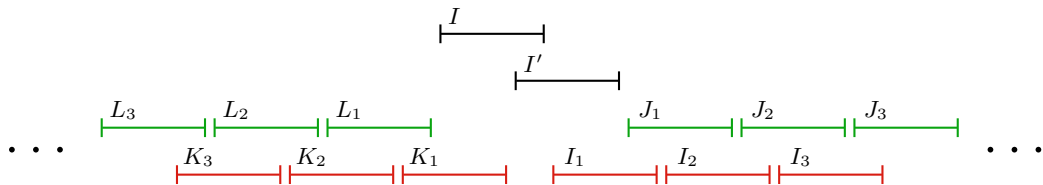
Since every block is assigned at most $(k + 1)$ consecutive ranks, if we set the jump number $\kappa = \mathcal{O}(k^4)$, the cost incurred in the Frogs game covers the ranks assigned to the $\mathcal{O}(k^3)$ consecutive blocks, which in turn bound the recoloring cost in the recoloring algorithm for step j . Of course, upon the arrival of I_j some cliques need to be merged into one passive block, and even some passive blocks get merged together, meaning that we need to merge some extra ranks in the rank sequence, but this also can be handled by the FROGS game. ◀

3.4 Fully Dynamic Unit Interval Recoloring problem

In this section we show that for the FULLY DYNAMIC UNIT INTERVAL RECOLORING problem one cannot hope for an algorithm with a reasonably limited recourse budget. In the FULLY DYNAMIC UNIT INTERVAL RECOLORING problem, we are initially given an empty multiset of unit intervals $\mathcal{I}_0 = \emptyset$. The instance \mathcal{I}_{j+1} is obtained from \mathcal{I}_j by adding a new unit interval to \mathcal{I}_j or removing the existing chosen interval from \mathcal{I}_j . Every instance \mathcal{I}_j presented to the algorithm is k -colorable. The goal is again to maintain a proper k -coloring on the dynamic instance. After each interval insertion and removal, the algorithm needs to compute a proper k -coloring c_j for \mathcal{I}_j . Similarly as before, our objective is to minimize the recourse budget, which is the number of intervals with different colors in c_j and c_{j-1} . For this problem we get the following negative result.

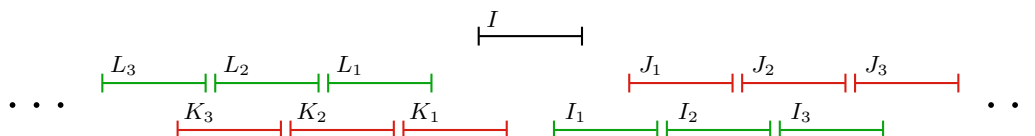
► **Observation 10.** *There is a sequence of m updates for the FULLY DYNAMIC UNIT INTERVAL RECOLORING problem that forces the total number of recolorings of $\Omega(m^2)$.*

Proof. We first construct an instance $\mathcal{M} = \mathcal{I} \cup \mathcal{J} \cup \mathcal{K} \cup \mathcal{L}$, where $\mathcal{I} = \{I_1, \dots, I_n\}$, $\mathcal{J} = \{J_1, \dots, J_n\}$, $\mathcal{K} = \{K_1, \dots, K_n\}$ and $\mathcal{L} = \{L_1, \dots, L_n\}$. All four sets contain pairwise disjoint intervals. Both $\mathcal{K} \cup \mathcal{L}$ and $\mathcal{I} \cup \mathcal{J}$ are paths when interpreted as graphs. Additionally, set $\mathcal{K} \cup \mathcal{L}$ is placed to the left of $\mathcal{I} \cup \mathcal{J}$. This is shown in Figure 5.



■ **Figure 5** The update when K_1 has the same color as I_1 .

Observe that since \mathcal{M} is 2-colorable, all sets \mathcal{K} , \mathcal{L} , \mathcal{I} and \mathcal{J} are necessarily monochromatic, moreover set \mathcal{K} has different color than \mathcal{L} , and \mathcal{I} has different color than \mathcal{J} . We construct the instance so, that K_1 is the rightmost interval of $\mathcal{K} \cup \mathcal{L}$ and interval I_1 is the leftmost interval of $\mathcal{I} \cup \mathcal{J}$. Moreover, the distance between the end coordinate of K_1 and the begin coordinate of I_1 is less than one (see Figure 5). Consider the case when K_1 has the same color as I_1 . Then we insert to \mathcal{M} two intersecting intervals I and I' , such that I intersects K_1 and I' intersects I_1 (see Figure 5). This results in $\mathcal{M} \cup \{I, I'\}$ being a path when interpreted as a graph. The instance is still 2-colorable, but now \mathcal{I} needs to have the same color as \mathcal{L} and \mathcal{J} needs to have the same color as \mathcal{K} . This requires recoloring $2n$ intervals, since either $\mathcal{I} \cup \mathcal{J}$ or $\mathcal{K} \cup \mathcal{L}$ has to be recolored. Next, we remove I and I' . Consider now the case when K_1 has a different color than I_1 (see Figure 6). In that case we insert an interval I intersecting



■ **Figure 6** The update when K_1 has different color than I_1 .

both K_1 and I_1 . This causes that K_1 and I_1 have to now be assigned different colors, and again either $\mathcal{I} \cup \mathcal{J}$ or $\mathcal{K} \cup \mathcal{L}$ has to be recolored. ◀

3.5 Coloring Unit Circular Arc Graphs

In this section we present a different application of the Precoloring Extension Lemma (Lemma 5). As a result, we offer a new combinatorial result for coloring unit circular arc graphs.

Unit circular arc graphs, as a subclass of proper circular arc graphs, admit an $\mathcal{O}(n^{1.5})$ algorithm that statically finds an optimum proper coloring [28]. On the other hand, the problem of coloring circular arc graphs is NP-hard [16], and a lot of research has been devoted to find positive results regarding this problem. This line of research exposes two important parameters describing an instance \mathcal{A} of a circular arc graph: the load $\text{load}(\mathcal{A})$ and the cover number $\text{cn}(\mathcal{A})$. The load $\text{load}(\mathcal{A})$ stands for the maximum number of arcs intersecting in one point, whereas the cover number $\text{cn}(\mathcal{A})$ stands for the minimum number of arcs covering the whole circle. The research focus is on the conditions under which a circular arc graph admits a proper coloring using close to $\text{load}(\mathcal{A})$ colors. Tucker [30] shows, that if $\text{cn}(\mathcal{A}) \geq 4$, then $\lfloor 3 \text{load}(\mathcal{A}) / 2 \rfloor$ colors suffice. Valencia-Pabon [31] shows that if $\text{cn}(\mathcal{A}) \geq 5$, then $\left\lceil \frac{\text{cn}(\mathcal{A}) - 1}{\text{cn}(\mathcal{A}) - 2} \text{load}(\mathcal{A}) \right\rceil$ colors is enough. For $\text{cn}(\mathcal{A}) \geq \text{load}(\mathcal{A}) + 2$ the bound becomes $\text{load}(\mathcal{A}) + 1$. Belkale and Chandran [4] prove the Hadwiger’s conjecture for proper circular arc graphs. Neither of these results exposes a condition sufficient to color the instance with precisely $\text{load}(\mathcal{A})$ colors. We use Lemma 5 to show, that if one can add $\text{load}(\mathcal{A})^2 - 1$ non-intersecting unit arcs to an instance of a unit circular arc graph in a way that the load does not increase, then $\text{load}(\mathcal{A})$ colors is sufficient to properly color the instance. This is formalized by the following lemma, proved in the full version [11] in Appendix D.

► **Theorem 11.** *Let \mathcal{A} be a set of unit circular-arcs on the circle with a circumference at least 2, such that \mathcal{A} can be extended with $r = \text{load}(\mathcal{A})^2 - 1$ not intersecting unit circular-arcs B_1, \dots, B_r which do not increase the load, i.e., $\text{load}(\mathcal{A}) = \text{load}(\mathcal{A} \cup \{B_1, \dots, B_r\})$. Then $\chi(\mathcal{A}) \leq \text{load}(\mathcal{A})$.*

Note that our condition of Theorem 11 can be easily checked in linear time and it might be a very natural assumption for some applications.

4 Concluding remarks

In this section we discuss several interesting extensions and aspects of our main result, which is the recoloring algorithm for the INCREMENTAL UNIT INTERVAL RECOLORING problem. We conclude with future research directions and open problems.

The first interesting property of the RECOLOR algorithm is its resistance to malicious coloring. By that we mean, that the RECOLOR algorithm on its input coloring does not assume anything other than being a proper k -coloring. In other words, some evil adversary could potentially repaint the whole instance before inserting the new interval, and as long as this is a proper k -coloring, the RECOLOR algorithm works. This might be of interest for some applications, one of which is described next.

Throughout the paper we make a simplifying assumption that the chromatic number k of the final instance is given apriori to the RECOLOR algorithm (as a parameter). Using the fact that our algorithm works against malicious coloring, it is easy to get rid of this assumption by running the recoloring algorithm with changing values of k . To be more precise, let us consider each step j when the chromatic number increases: $l - 1 = \chi(\mathcal{I}_{j-1}) < \chi(\mathcal{I}_j) = l$ (we can easily detect all such steps j). Before I_j arrives, we have the $(l - 1)$ -coloring c_{j-1} of \mathcal{I}_{j-1} at our disposal. Coloring c is obviously also an l -coloring for \mathcal{I}_{j-1} , so the recoloring algorithm for \mathcal{I}_j gets the correct input. From now on, until the next step when the chromatic number increases, the algorithm runs with parameter $k = l$. It is easy to see that the algorithm modified in this way returns the proper coloring. Let K be the chromatic number of the final instance \mathcal{I}_n (not known to the algorithm). Since for every k used by the modified algorithm we have $k \leq K$, the analysis for parameter K bounds from above the total recoloring budget of the modified algorithm. Thus, using $\mathcal{O}(K^7 \log n)$ amortized recoloring budget, we can maintain an optimal coloring for each instance \mathcal{I}_j .

Our incremental algorithm can be implemented in total time $\mathcal{O}(k^7 n \log n)$, where k is the final chromatic number. It suffices to maintain a sorted list of intervals \mathcal{I}_j in an AVL tree. Such a list allows inserting a new interval in time $\mathcal{O}(\log n)$. It allows finding the predecessor and the successor of the newly inserted interval, and efficient iteration to the left and to the right. This allows detecting the infix that we want to recolor in the time proportional to the size of the infix. The coloring step can then be performed in linear time. We refer to the full version [11] of the paper for the details and the pseudocode of the algorithm.

Finally, let us shortly discuss why our approach does not seem to extend to general interval graphs, or even the intervals whose lengths vary from 1 to $(1 + \epsilon)$. The main reason for that is that the MODULO COLOR COMPLETION algorithm spectacularly fails on such graphs. In particular, Observation 2 ceases to hold, and the MODULO COLOR COMPLETION algorithm is based on this observation. It would be interesting to see if the color sorting technique could work with some algorithms other than MODULO COLOR COMPLETION, effective on any superclass of unit interval graphs. We leave this as the main open question. Note that due to our lower bound in Section 3.4, which carries over to any superclass of unit interval graphs, we cannot hope on positive results regarding the fully dynamic setting.

References

- 1 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPICs*, pages 41:1–41:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.41.

- 2 Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. In *Algorithms and data structures*, volume 10389 of *Lecture Notes in Comput. Sci.*, pages 97–108. Springer, Cham, 2017. doi:10.1007/978-3-319-62127-2_9.
- 3 Dwight R. Bean. Effective coloration. *The Journal of Symbolic Logic*, 41(2):469–480, 1976. URL: <http://www.jstor.org/stable/2272247>.
- 4 Naveen Belkale and L. Sunil Chandran. Hadwiger’s conjecture for proper circular arc graphs. *European J. Combin.*, 30(4):946–956, 2009. doi:10.1016/j.ejc.2008.07.024.
- 5 Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized $O(\log^2 n)$ replacements. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 947–959. SIAM, Philadelphia, PA, 2018. doi:10.1137/1.9781611975031.61.
- 6 Bartłomiej Bosek, Stefan Felsner, Kamil Kloch, Tomasz Krawczyk, Grzegorz Matecki, and Piotr Micek. On-line chain partitions of orders: a survey. *Order*, 29(1):49–73, 2012. doi:10.1007/s11083-011-9197-1.
- 7 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *55th Annual IEEE Symposium on Foundations of Computer Science – FOCS 2014*, pages 384–393. IEEE Computer Soc., Los Alamitos, CA, 2014. doi:10.1109/FOCS.2014.48.
- 8 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. Shortest augmenting paths for online matchings on trees. *Theory Comput. Syst.*, 62(2):337–348, 2018. doi:10.1007/s00224-017-9838-x.
- 9 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. A tight bound for shortest augmenting paths on trees. In *LATIN 2018: Theoretical informatics*, volume 10807 of *Lecture Notes in Comput. Sci.*, pages 201–216. Springer, Cham, 2018. doi:10.1007/978-3-319-77404-6_1.
- 10 Bartłomiej Bosek, Yann Disser, Andreas Emil Feldmann, Jakub Pawlewicz, and Anna Zych-Pawlewicz. Recoloring Interval Graphs with Limited Recourse Budget. In Susanne Albers, editor, *17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)*, volume 162 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:23, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2020.17.
- 11 Bartłomiej Bosek and Anna Zych-Pawlewicz. Recoloring unit interval graphs with logarithmic recourse budget, 2022. doi:10.48550/ARXIV.2202.08006.
- 12 Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski, and Anna Zych-Pawlewicz. *Efficient fully dynamic elimination forests with applications to detecting long paths and cycles*, pages 796–809. SIAM, 2021. doi:10.1137/1.9781611976465.50.
- 13 Mitre Costa Dourado, Van Bang Le, Fábio Protti, Dieter Rautenbach, and Jayme Luiz Swarcfiter. Mixed unit interval graphs. *Discret. Math.*, 312:3357–3363, 2012.
- 14 Zdeněk Dvořák, Martin Kupec, and Vojtěch Tůma. A dynamic data structure for MSO properties in graphs with bounded tree-depth. In *Proceedings of the 22nd Annual European Symposium on Algorithms, ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2014. doi:10.1007/978-3-662-44777-2_28.
- 15 Leah Epstein and Meital Levy. Online interval coloring and variants. In *Proceedings of the 32nd International Conference on Automata, Languages and Programming, ICALP’05*, pages 602–613, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11523468_49.
- 16 M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic Discrete Methods*, 1(2):216–227, 1980. doi:10.1137/0601025.

- 17 Martin Charles Golumbic. Chapter 8 – interval graphs. In Martin Charles Golumbic, editor, *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*, pages 171–202. Elsevier, 2004. doi:10.1016/S0167-5060(04)80056-6.
- 18 Magnús M. Halldórsson and Mario Szegedy. Lower bounds for on-line graph coloring, 1994.
- 19 Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991. doi:10.1137/0404033.
- 20 Henry A. Kierstead and William T. Trotter, Jr. An extremal problem in recursive combinatorics. *Congr. Numer.*, 33:143–153, 1981.
- 21 Jakub Łącki, Jakub Oćwieja, Marcin Pilipczuk, Piotr Sankowski, and Anna Zych. The power of dynamic distance oracles: efficient dynamic algorithms for the Steiner tree. In *STOC'15 – Proceedings of the 2015 ACM Symposium on Theory of Computing*, pages 11–20. ACM, New York, 2015.
- 22 Dániel Marx. Precoloring extension on unit interval graphs. *Discrete Applied Mathematics*, 154(6):995–1002, 2006. doi:10.1016/j.dam.2005.10.008.
- 23 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. In *Automata, languages, and programming. Part I*, volume 7391 of *Lecture Notes in Comput. Sci.*, pages 689–700. Springer, Heidelberg, 2012. doi:10.1007/978-3-642-31594-7_58.
- 24 Stephan Olariu. An optimal greedy heuristic to color interval graphs. *Inform. Process. Lett.*, 37(1):21–25, 1991. doi:10.1016/0020-0190(91)90245-D.
- 25 James B. Orlin, Maurizio A. Bonuccelli, and Daniel P. Bovet. An $O(n^2)$ algorithm for coloring proper circular arc graphs. *SIAM Journal on Algebraic Discrete Methods*, 2(2):88–93, 1981. doi:10.1137/0602012.
- 26 H. Maehara P. Frankl. Open-interval graphs versus closed-interval graphs. *Discret. Math.*, 63:97–100, 1987.
- 27 Rasmus V. Rasmussen and Michael A. Trick. Round robin scheduling – a survey. *European Journal of Operational Research*, 188(3):617–636, 2008. doi:10.1016/j.ejor.2007.05.046.
- 28 Wei-Kuan Shih and Wen-Lian Hsu. An $o(n^{1.5})$ algorithm to color proper circular arcs. *Discrete Applied Mathematics*, 25(3):321–323, 1989. doi:10.1016/0166-218X(89)90011-5.
- 29 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. In *26th European Symposium on Algorithms*, volume 112 of *LIPICs. Leibniz Int. Proc. Inform.*, pages Art. No. 72, 16. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.
- 30 Alan Tucker. Coloring a family of circular arcs. *SIAM Journal on Applied Mathematics*, 29(3):493–502, 1975. doi:10.1137/0129040.
- 31 Mario Valencia-Pabon. Revisiting Tucker’s algorithm to color circular arc graphs. *SIAM Journal on Computing*, 32(4):1067–1072, 2003. doi:10.1137/S0097539700382157.

Polynomial Kernel for Immersion Hitting in Tournaments

Łukasz Bożyk  

University of Warsaw, Poland

Michał Pilipczuk  

University of Warsaw, Poland

Abstract

For a fixed simple digraph H without isolated vertices, we consider the problem of deleting arcs from a given tournament to get a digraph which does not contain H as an immersion. We prove that for every H , this problem admits a polynomial kernel when parameterized by the number of deleted arcs. The degree of the bound on the kernel size depends on H .

2012 ACM Subject Classification Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases kernelization, graph immersion, tournament, protrusion

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.26

Related Version *Full Version*: <http://arxiv.org/abs/2208.07789>

Funding This work is a part of projects that have received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme, grant agreements No. 714704 (Ł. Bożyk) and No. 948057 (Mi. Pilipczuk).



1 Introduction

Kernelization is an algorithmic framework for describing preprocessing procedures that given an instance of a hard problem, identify and reduce easily resolvable parts. The usual formalization of the concept is based on the paradigm of parameterized complexity. A *kernelization procedure*, or a *kernel* for short, for a parameterized decision problem L is a polynomial-time algorithm that given an instance (I, k) of L , where k is the parameter, outputs an equivalent instance (I', k') such that both $|I'|$ and k' are bounded by a computable function of k . If this function is a polynomial in k , we say that the kernel is *polynomial*. The search for (polynomial) kernels is an established research area within the field of parameterized algorithms. We refer to the textbook of Fomin et al. [13] for a broader discussion of classic results and techniques.

A particularly fruitful line of research within kernelization concerns the methodology of *protrusions* and *protrusion replacement*. The idea is to find a large *protrusion*: a piece of graph that is simple – for instance, has bounded treewidth – and communicates with the rest of the graph only through a small interface. If found, a protrusion can be fully understood – for instance, using dynamic programming on its tree decomposition – and replaced with a smaller one with the same functionality. So if one proves that, provided the given instance is large, a large protrusion can be efficiently found and replaced with a strictly smaller one, then applying this strategy exhaustively eventually arrives at a kernel. Protrusion-based techniques were pioneered by Bodlaender et al. [5], but by now have become a part of the standard toolbox of kernelization. We refer the interested reader to [13, Part 2] for more information.

A particularly important achievement in the development of protrusion-based kernelization procedures is the result of Fomin et al. [12], who gave a polynomial kernel for the PLANAR \mathcal{F} -DELETION problem, defined as follows. Let \mathcal{F} be a fixed family of graphs containing



© Łukasz Bożyk and Michał Pilipczuk;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 26; pp. 26:1–26:17

Leibniz International Proceedings in Informatics



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

at least one planar graph. Then in the problem we are given a graph G and an integer k (considered to be the parameter), and the question is whether one can hit all minor models of graphs from \mathcal{F} in G using a hitting set consisting of at most k vertices. Fomin et al. gave a polynomial kernel for this problem for every fixed family \mathcal{F} as above. The degree of the polynomial bound on the kernel size depends on \mathcal{F} and it is known that under certain complexity-theoretical assumptions, this is unavoidable [15]. The assumption that \mathcal{F} contains a planar graph is crucial in the approach: under this assumption, graphs not containing any graph from \mathcal{F} as a minor have treewidth bounded by a constant, which unlocks a multitude of tools related to tree decomposition using which one can understand the structure of the instance.

The concept of a protrusion, as described above, is quite capacious and can be applied in different settings as well. For instance, Giannopolou et al. [18] considered the \mathcal{F} -IMMERSION DELETION problem, where for a given graph G and parameter k , one wishes to hit all immersion models of graphs from \mathcal{F} using a hitting set of edges of size at most k (Recall that an *immersion model* of a graph H in a graph G consists of mapping the vertices of H to distinct vertices of G and edges of H to pairwise edge-disjoint paths in G so that the image of an edge uv connects the image of u with the image of v). By loosely following the approach of [12], Giannopolou et al. [18] gave a *linear* kernel for \mathcal{F} -IMMERSION DELETION for every family \mathcal{F} that contains a subcubic planar graph. Here, the main idea was to adjust the notions of protrusions to the graph parameter *tree-cutwidth* and corresponding *tree-cut decompositions*, which play the same role for immersions as treewidth and tree decompositions play for minors.

Motivated by this, it is interesting to consider other settings where the protrusion methodology could be applied. A particularly tempting area is that of *directed graphs*, where natural analogues of problems considered in undirected graphs can be easily stated. Unfortunately, the structural theory of directed graphs is much less understood than that of undirected graphs, and many problems become inherently harder; see e.g. [16, 17, 20]. In particular, there is even a scarcity of fixed-parameter tractability results, not to mention kernelization results.

However, there is a particular class of directed graphs where a sound structural theory has been developed: *tournaments*. (Here, recall that a *tournament* is a directed graph where every pair of vertices is connected by exactly one arc.) This theory¹ was pioneered by Chudnovsky, Ovetsky, Fradkin, Kim, and Seymour [9, 10, 26, 27, 21], while structural and algorithmic aspects connected to parameterized complexity were investigated by Fomin and Pilipczuk [14]. See the introductory section of [14] for an overview.

In particular, as proved in the aforementioned works, there are two main width notions for tournaments: *cutwidth* and *pathwidth*. The first one is tightly connected to (directed) immersions as follows: if a tournament T excludes a digraph H as an immersion, then the cutwidth of T is bounded by a constant depending only on H . Pathwidth is connected to *topological minors* and *strong minors* in the same sense. These structural results were used for the design of parameterized algorithms for containment problems in tournaments in [9, 26, 14]. Later, they were used by Raymond [28] and by Bożyk and Pilipczuk [8] to establish Erdős-Pósa properties for immersions and topological minors in tournaments.

The goal of this work is to explore the applicability of the structural theory of tournaments for kernelization, with a particular focus of developing a sound protrusion-based methodology.

¹ In this line of work, most results concern the class of *semi-complete digraphs*, which differ from tournaments by allowing that a pair of vertices can be also connected by two oppositely-oriented arcs. In this article we focus on the setting of tournaments for simplicity.

Our contribution. For a simple directed graph H without isolated vertices we define the following parameterized problem:

H -HITTING IMMERSIONS IN TOURNAMENTS

Input: A tournament T and a positive integer k .

Parameter: k

Output: Is there a set $F \subseteq A(T)$, such that $|F| \leq k$ and $T - F$ is H -immersion-free?

That this problem is fixed-parameter tractable is proved in [14]. Our main result states that for every fixed H , H -HITTING IMMERSIONS IN TOURNAMENTS admits a polynomial kernel, of degree dependent on H . Formally, we prove the following theorem.

► **Theorem 1.** *For every simple digraph H without isolated vertices there exists a constant c and an algorithm that given an instance (T, k) of H -HITTING IMMERSIONS IN TOURNAMENTS, runs in polynomial time and returns an equivalent instance (T', k) with $|T'| \leq c \cdot k^c$.*

We remark that when H is the directed triangle, H -HITTING IMMERSIONS IN TOURNAMENTS is equivalent to the FEEDBACK ARC SET IN TOURNAMENTS (FAST) problem. There is a sizeable literature on the parameterized complexity of FAST, see e.g. [1, 11, 14, 19], mostly due to the fact that it admits a subexponential parameterized algorithm with running time $2^{\mathcal{O}(\sqrt{k})} \cdot n^{\mathcal{O}(1)}$. Kernelization procedures for FAST were investigated by Bessy et al. in [4], while kernelization of the dual problems of packing arc-disjoint triangles and packing arc-disjoint cycles in tournaments were recently studied by Bessy et al. in [3].

On a very high conceptual level, the proof of Theorem 1 follows the classic blueprint of protrusion-based kernelization, like in e.g. [12, 18]. That is, if (T, k) is a given instance of H -HITTING IMMERSIONS IN TOURNAMENTS, we perform the following steps.

- We may assume that the cutwidth of T is bounded polynomially in k , for otherwise in T one can find $k + 1$ arc-disjoint immersion models of H ; these witness a negative answer to the instance.
- Assuming that T is large – of size superpolynomial in k – but has cutwidth bounded polynomially in k , we may find in T a large *protrusion*. Here, a protrusion is an interval I in the vertex ordering σ witnessing small cutwidth such that σ restricted to I witnesses that $T[I]$ has constant cutwidth, and there is only a constant number of σ -backward arcs with one endpoint in I and the other outside of I . These are instantiations of the two desired properties of a protrusion: it has to have bounded width and communicate with the rest of the graph through a boundary of bounded size.
- We can replace the obtained protrusion with a strictly smaller one of the same “type”, thus obtaining a strictly smaller equivalent instance. Applying this strategy exhaustively eventually yields a kernel of polynomial size.

Compared to the previous works, the main difficulty is to tame the interaction between a protrusion and the remainder of the instance. Namely, this interaction is not restricted to a set of vertices or arcs of constant size: as we work with tournaments, every vertex of a protrusion will necessarily have an arc connecting it to every vertex outside of the protrusion. The idea is that all but a constant number of those arcs will be forward arcs in the fixed vertex ordering σ with bounded cutwidth. We call those well-behaved forward arcs *generic*, while the remaining constantly many backward arcs are *singular*. Understanding the interaction between a protrusion and the rest of the tournament as being governed by few singular arcs and a large number of well-behaved generic arcs is the crux of our approach.

In particular, while looking for a large replaceable protrusion, we have to be extremely careful when arguing about how such a protrusion may interact with optimum solutions. Here, a key step is to find several protrusions that appear consecutively in σ (recall that our protrusions are intervals in σ), have the same *type* (in the sense of admitting partial immersions of H), and such that their union is a protrusion of again the same type. This step is done using Simon Factorization, a tool commonly used in the theory of automata and formal languages. Simon Factorization was recently used a few times in structural graph theory [7, 23, 24], but we are not aware of any previous application in the context of kernelization.

The application of Simon Factorization is also the only step in the reasoning that causes the degree of the polynomial bounding the size of our kernel to depend on H . It is an interesting open question whether this can be improved, or in other words, whether there is a kernel of size at most $c \cdot k^d$, where c may depend on H but d does not. Judging by the results on hitting immersions in undirected graphs [18], we expect that this might be the case.

We remark that the sign \asymp denotes statements whose proofs are deferred to the full version of the paper.

2 Preliminaries

We use the standard terminology and notation for describing immersions in tournaments and for cutwidth of digraphs and of tournaments. This terminology and notation is borrowed mostly, and often in a verbatim form, from the work of Bożyk and Pilipczuk [8].

For a positive integer n , we denote $[n] := \{1, \dots, n\}$ and $[-n] = \{-1, \dots, -n\}$.

We use standard graph terminology and notation. All graphs considered in this paper are finite, simple (i.e. without self-loops or multiple arcs with same head and tail), and directed (i.e. are *digraphs*). For a digraph D , by $V(D)$ and $A(D)$ we denote the vertex set and the arc set of D , respectively. We denote

$$|D| := |V(D)| \quad \text{and} \quad \|D\| := |A(D)|.$$

For $X \subseteq V(D)$, the subgraph *induced* by X , denoted $D[X]$, comprises of the vertices of X and all the arcs of D with both endpoints in X . By $D - X$ we denote the digraph $D[V(D) \setminus X]$. Further, if F is a subset of arcs of D , then by $D - F$ we denote the digraph obtained from D by removing all the arcs of F . For $X, Y \subseteq V(D)$ we denote by $\vec{A}(X, Y)$ the set of all arcs $(u, v) \in A(D)$ such that $u \in X$ and $v \in Y$ and moreover $A(X, Y) := \vec{A}(X, Y) \cup \vec{A}(Y, X)$. For an arc $a = (u, v) \in A(D)$ we define $\text{tail}(a) := u$ and $\text{head}(a) := v$. For a directed (not necessarily simple) path P we denote by $\text{first}(P)$ and $\text{last}(P)$ the first and the last arcs on path P , respectively.

Tournaments. A simple digraph $T = (V, A)$ is called a *tournament* if for every pair of distinct vertices $u, v \in V$, either $(u, v) \in A$, or $(v, u) \in A$ (but not both). Alternatively, one can represent the tournament T by providing a pair $(\sigma, \overleftarrow{A}_\sigma(T))$, where $\sigma: V \rightarrow [|V|]$ is an *ordering* of the set V and $\overleftarrow{A}_\sigma(T)$ is the set of σ -backward arcs, that is,

$$\overleftarrow{A}_\sigma(T) := \{ (u, v) \in A \mid \sigma(u) > \sigma(v) \}.$$

All the remaining arcs are called σ -forward. If the choice of ordering σ is clear from the context, we will call the arcs simply *backward* or *forward*.

Cutwidth. Let $T = (V, A)$ be a tournament and σ be an ordering of V . For $\alpha, \beta \in \{0, 1, \dots, |V|\}$, $\alpha \leq \beta$, we define

$$\sigma(\alpha, \beta] := \{v \in V \mid \alpha < \sigma(v) \leq \beta\}.$$

Sets $\sigma(\alpha, \beta]$ as defined above will be called σ -intervals. The set

$$\text{cut}_\sigma[\alpha] = \{(u, v) \in A \mid \sigma(u) > \alpha \geq \sigma(v)\} \subseteq \overleftarrow{A}_\sigma(T)$$

is called the α -cut of σ . The *width* of the ordering σ is equal to $\max_{0 \leq \alpha \leq |V|} |\text{cut}[\alpha]|$, and the *cutwidth* of T , denoted $\text{ctw}(T)$, is the minimum width among all orderings of V .

It is perhaps a bit surprising that in tournaments, there is a very simple algorithm to compute an ordering of optimum width: just sort the vertices by outdegrees.

► **Lemma 2** (see [2, 25]). *Let T be a tournament and σ be an ordering of T satisfying the following for every pair of vertices u and v : if u appears before v in σ , then the outdegree of u is not smaller than the outdegree of v . Then the width of σ is equal to $\text{ctw}(T)$.*

If $I = \sigma(\alpha, \beta]$, then we denote

$$\partial^+(I) := \overrightarrow{A}(I, \sigma(0, \alpha]) \quad \text{and} \quad \partial^-(I) := \overrightarrow{A}(\sigma(\beta, |V|], I).$$

Note that $\partial^+(I) \subseteq \text{cut}_\sigma[\alpha]$ and $\partial^-(I) \subseteq \text{cut}_\sigma[\beta]$ and therefore $|\partial^+(I)| \leq c$ and $|\partial^-(I)| \leq c$, where c is the width of σ . These inclusions may be proper, as the arcs from the set $\widehat{\partial}(I) := \overrightarrow{A}(\sigma(\beta, |V|], \sigma(0, \alpha])$ contribute to the cuts but are not incident with I . We define $\partial(I) := \partial^+(I) \cup \partial^-(I)$ and call the elements of $\partial(I)$ *I-singular* (or simply *singular*) arcs. Moreover, we define

$$\Gamma^+(I) := \overrightarrow{A}(I, \sigma(\beta, |V|]), \quad \Gamma^-(I) := \overrightarrow{A}(\sigma(0, \alpha], I), \quad \text{and} \quad \Gamma(I) = \Gamma^+(I) \cup \Gamma^-(I),$$

and call the elements of $\Gamma(I)$ *I-generic* (or simply *generic*) arcs.

If $I' = V - I$ where $I = \sigma(\alpha, \beta]$, then we call the set I' a *co-interval* and define *I'-singular* and *I'-generic* arcs as follows

$$\partial^-(I') := \partial^+(I), \quad \partial^+(I') := \partial^-(I), \quad \partial(I') := \partial^+(I') \cup \partial^-(I') = \partial(I),$$

$$\Gamma^-(I') := \Gamma^+(I), \quad \Gamma^+(I') := \Gamma^-(I), \quad \Gamma(I') := \Gamma^-(I') \cup \Gamma^+(I') = \Gamma(I).$$

Immersion. A digraph \widehat{H} is an *immersion model* (or briefly a *copy*) of a digraph H if there exists a mapping ϕ , called an *immersion embedding*, such that:

- vertices of H are mapped to pairwise different vertices of \widehat{H} ;
- each arc $(u, v) \in A(H)$ is mapped to a directed path in \widehat{H} starting at $\phi(u)$ and ending at $\phi(v)$; and
- each arc of \widehat{H} belongs to exactly one of the paths $\{\phi(a) : a \in A(H)\}$.

If it does not lead to misunderstanding, we will sometimes slightly abuse the above notation by identifying ϕ and \widehat{H} and calling ϕ the immersion model of H .

Let H be a digraph. We say that a digraph G *contains H as an immersion* (or H can be *immersed* in G) if G has a subgraph that is an immersion model of H . Digraph G is called *H-immersion-free* (or *H-free* for brevity) if it does not contain H as an immersion.

We will use the following result of Fomin and Pilipczuk [14].

► **Theorem 3** (Theorem 7.3 of [14]). *Let T be a tournament which does not contain a digraph H as an immersion. Then $\text{ctw}(T) \in \mathcal{O}((|H| + \|H\|)^2)$.*

► **Corollary 4.** *For every digraph H there exists a constant c_H such that for every H -free tournament T , we have $\text{ctw}(T) \leq c_H$.*

Throughout this paper we fix a simple digraph H without isolated vertices and an integer $k \in \mathbb{N}$. For a tournament T , a set $F \subseteq A(T)$ is called a *solution* if $T - F$ is H -free. Moreover, F is an *optimal solution* if it is a solution of the smallest possible size. So (T, k) is a YES-instance of H -HITTING IMMERSIONS IN TOURNAMENTS if and only if there exists an optimal solution in T of size at most k .

Monoids and Simon factorization. Simon factorization was originally developed by Simon in [29] and the currently best bounds are due to Kufleitner [22]. See also the work of Bojańczyk [6] for a nice exposition; we mostly follow the notation from that source.

Let S be a finite monoid (i.e., a finite set equipped with an associative binary operation \cdot and a neutral element 1). An element $e \in S$ is called *idempotent* if $e \cdot e = e$. For a finite alphabet A , by A^* we denote the set of all finite words over A , and a *morphism* $\alpha: A^* \rightarrow S$ is a function satisfying $\alpha(\varepsilon) = 1$ (ε being the empty word) and $\alpha(w_1 w_2) = \alpha(w_1) \cdot \alpha(w_2)$ for every $w_1, w_2 \in A^*$. Note that a morphism is uniquely defined by the images of single symbols from A . The following lemma is a direct consequence of Simon Factorization.

► **Lemma 5.** *Let S be a finite monoid, A be a finite alphabet, and $\alpha: A^* \rightarrow S$ be a morphism. Suppose $w \in A^*$ is a word of length at least $\ell^{3|S|}$. Then there exists a subword w' of w and an idempotent $e \in S$ such that $w' = w_1 w_2 \dots w_\ell$, where $w_i \in A^*$ are nonempty subwords of w and*

$$\alpha(w_1) = \alpha(w_2) = \dots = \alpha(w_\ell) = e.$$

Note that in the setting of Lemma 5, given a word $w \in A^*$ of length $n \geq \ell^{3|S|}$, one can easily find w' and a suitable decomposition $w' = w_1 w_2 \dots w_\ell$ in time $\mathcal{O}(|S| \cdot n^3)$ assuming unit cost of operations in S . Indeed, one can guess e (by trying at most $|S|$ possibilities) and the first position of w' within w (by trying n possibilities), and then for every subword w'' starting at this position compute the longest possible decomposition of the form $w'' = w_1 w_2 \dots w_\ell$ such that $\alpha(w_1) = \dots = \alpha(w_\ell) = e$, if existent. The latter can be done by a standard left-to-right dynamic programming in time $\mathcal{O}(n^2)$.

3 Partial immersions

Our goal in this section is to extend the notion of an immersion to *partial immersions*. These will be used to understand possible behaviors of immersion models in a tournament T with respect to different intervals in an ordering of the vertex set of T . Let then $T = (V, A)$ be a tournament and let σ be an ordering of V . For now, fix a σ -interval $I := \sigma(\alpha, \beta]$.

Partial immersions.

► **Definition 6.** *A scattered path in I of size $q \geq 0$ is a sequence $\tilde{P} = (P_i)_{i=1}^q$ satisfying the following properties:*

- *for each $i \in [q]$, P_i is a directed (simple) path of length at least 1 consisting of arcs that belong to $A(T[I]) \cup \partial(I) \cup \Gamma(I)$;*
- *paths P_i for $i \in [q]$ are pairwise arc-disjoint;*
- *for every $i \in [q]$, $i \neq 1$, we have $\text{first}(P_i) \in \Gamma^-(I) \cup \partial^-(I)$;*
- *for every $i \in [q]$, $i \neq q$, we have $\text{last}(P_i) \in \Gamma^+(I) \cup \partial^+(I)$.*

Each term in the sequence $(P_i)_{i=1}^q$ will be called a piece of \tilde{P} . The set of arcs of all pieces of \tilde{P} is denoted $A(\tilde{P})$. If $\text{first}(P_i) \in \Gamma^-(I)$ and $\text{last}(P_i) \in \Gamma^+$, then the piece P_i is called generic.

Note that $\text{first}(P_1)$ is allowed to be an arc in the set $A(T[I]) \cup \Gamma^+(I) \cup \partial^+(I)$. If it is such, we call the vertex $\text{tail}(\text{first}(P_1)) \in I$ the *beginning* of \tilde{P} and denote it by $\text{start}(\tilde{P})$. Similarly, if $\text{last}(P_q) \in A(T[I]) \cup \Gamma^-(I) \cup \partial^-(I)$, then we call the vertex $\text{head}(\text{last}(P_q))$ the *end* of \tilde{P} and denote it by $\text{end}(\tilde{P})$. Note that the empty sequence is a scattered path of size 0. Also, a scattered path with only one piece, whose both beginning and end exist, is just a path in $T[I]$. By \mathcal{P}_I we denote the family of all scattered paths in I .

We say that a scattered path $\tilde{P} = (P_i)_{i=1}^q$ in I can be *shortened* to a scattered path \tilde{P}' (or that \tilde{P}' is a *shortening* of \tilde{P}) if:

- $\text{start}(\tilde{P}) = \text{start}(\tilde{P}')$ and $\text{end}(\tilde{P}) = \text{end}(\tilde{P}')$ (meaning either equal or simultaneously undefined);
- for each piece P_s of \tilde{P}' there exist indices $i_s^- \leq i_s^+$ such that $\text{tail}(\text{first}(P_s)) = \text{tail}(\text{first}(P_{i_s^-}))$ and $\text{head}(\text{last}(P_s)) = \text{head}(\text{last}(P_{i_s^+}))$; and
- whenever $s < s'$, we have $i_s^+ < i_{s'}^-$ and P_s appears before $P_{s'}$ in \tilde{P}' .

Intuitively, shortening of the path means removing some pieces and replacing several contiguous subsequences of the pieces with single pieces, keeping the tail of the beginning and the head of the end of the replaced subsequence. Note that in particular, some pieces of \tilde{P} can be simply omitted in \tilde{P}' (other than the initial and the terminal one).

► **Definition 7.** A partial immersion embedding of H in I (or shortly, a partial immersion in I) is a mapping $\phi: A(H) \rightarrow \mathcal{P}_I$ such that

- all scattered paths $\phi(a)$ for $a \in A(H)$ are pairwise arc-disjoint;
- if $\text{tail}(a) = \text{tail}(a')$ then $\text{start}(\phi(a))$ and $\text{start}(\phi(a'))$ are either equal, or simultaneously undefined;
- if $\text{start}(\phi(a))$ and $\text{start}(\phi(a'))$ are defined and equal, then $\text{tail}(a) = \text{tail}(a')$;
- if $\text{head}(a) = \text{head}(a')$ then $\text{end}(\phi(a))$ and $\text{end}(\phi(a'))$ are either equal, or simultaneously undefined;
- if $\text{end}(\phi(a))$ and $\text{end}(\phi(a'))$ are defined and equal, then $\text{head}(a) = \text{head}(a')$.

Intuitively, we can think of a partial immersion as of a “trace” which some immersion model \hat{H} of H in T leaves on the interval I . Some edges of H have images being paths in \hat{H} non-incident with I (these correspond to empty scattered paths in the partial immersion embedding). Some images of arcs of H come back and forth to I , intersecting with I along a non-empty scattered path (the ordering of paths on a single scattered path corresponds to the order of their appearance along the image of the respective arc of H). Finally, some arc images begin or end within I , which corresponds to the case when the beginning or the end of a scattered path is defined and is a vertex of I .

We call a partial immersion ϕ' in I a *shortening* of ϕ in I if for every $a \in A(H)$, the scattered path $\phi'(a)$ is a shortening of $\phi(a)$. We call ϕ *minimal* if there is no shortening of ϕ with at least one scattered path of strictly smaller size. Note that ϕ may be minimal even if some piece of some $\phi(a)$ can be replaced by a different single piece with equal first and last vertices. Shortening which does not decrease the size of any scattered path will be called *trivial*.

Note that each immersion model ϕ of H in T is a partial immersion in $V(T)$, in which all scattered paths $\phi(a)$, $a \in A(H)$, are paths in T beginning and ending at $\phi(\text{tail}(a))$ and $\phi(\text{head}(a))$, respectively. Moreover, each partial immersion ϕ in I gives rise to a natural partial immersion of H in $J \subseteq I$ in which all paths $\phi(a)$ where $a \in A(H)$ are “trimmed” to scattered paths consisting of precisely those arcs which are incident with J .

Formally, let I and J be σ -intervals such that $J \subseteq I$. If P is a path in I , then define the *trace* $P|_J$ of P on J to be the scattered path consisting of all arcs of P incident with J , arranged in the order of appearance along P . If $\tilde{P} = (P_i)_{i=1}^q$ is a scattered path in I , then

define the *trace* $\tilde{P}|_J$ of \tilde{P} on J to be the concatenation of scattered paths $P_i|_J$. The *trace* $\phi|_J$ of a partial immersion ϕ of H in I on J is defined by setting $\phi|_J(a) = (\phi(a))|_J$ for every $a \in A(H)$.

Consider any σ -intervals I_1, I_2 with the property that there exist $\alpha, \beta, \gamma \in \{0, 1, \dots, |V|\}$ such that $I_1 = \sigma(\alpha, \gamma]$ and $I_2 = \sigma(\gamma, \beta]$. We will call such a pair of intervals *consecutive*. Equivalently, two intervals are consecutive if their union $I = I_1 \cup I_2$ is a σ -interval as well. Let \tilde{P}_1 be a scattered path in I_1 and \tilde{P}_2 be a scattered path in I_2 . We say that \tilde{P}_1 and \tilde{P}_2 are *compatible* if

- $A(\tilde{P}_1) \cap A(I_1, I_2) = A(\tilde{P}_2) \cap A(I_1, I_2)$;
- the set of pieces whose arc set is $A(\tilde{P}_1) \cup A(\tilde{P}_2)$ can be ordered to form a scattered path \tilde{P} in I with the property that all pieces of \tilde{P}_i appear in \tilde{P} in the same order as they do in \tilde{P}_i , for $i = 1, 2$.

Every \tilde{P} described above will be called a *gluing* of \tilde{P}_1 and \tilde{P}_2 . Note that a gluing is not necessarily uniquely defined, which can be seen particularly well when $A(\tilde{P}_1) \cap A(I_1, I_2) = A(\tilde{P}_2) \cap A(I_1, I_2) = \emptyset$ – in this case the pieces of both paths can be “shuffled” in any way only keeping the order of pieces originating from the same path.

► **Observation 8.** *If \tilde{P}_1 is compatible with \tilde{P}_2 and \tilde{P}'_2 is a shortening of \tilde{P}_2 , then there exists a shortening \tilde{P}'_1 of \tilde{P}_1 such that \tilde{P}'_1 and \tilde{P}'_2 are compatible and every gluing of them is a shortening of some gluing of \tilde{P}_1 and \tilde{P}_2 .*

Proof. To construct \tilde{P}'_1 from \tilde{P}_1 it is enough to omit the pieces which do not share arcs with \tilde{P}'_2 . ◀

We will say that two partial immersions ϕ_1 in I_1 and ϕ_2 in I_2 are *compatible* if there exists a partial immersion ϕ in I such that $\phi_1 = \phi|_{I_1}$ and $\phi_2 = \phi|_{I_2}$, or – in other words – that for every $a \in A(H)$ the scattered path $\phi(a)$ is a gluing of $\phi_1(a)$ and $\phi_2(a)$. We will call every such ϕ a *gluing* of ϕ_1 and ϕ_2 . Note that gluing is not necessarily uniquely defined. Denote the set of all gluings of ϕ_1 and ϕ_2 by $\phi_1 \oplus \phi_2$.

► **Observation 9** (\times). *If $\phi \in \phi_1 \oplus \phi_2$ and ϕ is minimal, then ϕ_1 and ϕ_2 are minimal.*

The notions of a scattered path, partial immersion and trace can be naturally extended to co-intervals, by applying all definitions verbatim. If I is a σ -interval and $I' = V - I$ is the corresponding co-interval, then partial immersions ϕ_1 in I and ϕ_2 in I' are *compatible* if there exists an immersion ϕ in T such that $\phi_1 = \phi|_I$ and $\phi_2 = \phi|_{I'}$. Again, every such ϕ is called a *gluing* of ϕ_1 and ϕ_2 .

Types of intervals. The key ingredient of our analysis is a constant-size encoding of the set of possible “behaviors” of partial immersions in intervals.

A σ -interval I shall be called ℓ -*long* if $|I| \geq \ell$. Further, we shall call I c -*flat* if $|\partial^+(I)| \leq c$, $|\partial^-(I)| \leq c$, and σ restricted to $T[I]$ has width at most c .

Note that if $I = \sigma(\alpha, \beta]$ is $2r$ -long, then the intervals $I_r^- := \sigma(\alpha, \alpha+r]$ and $I_r^+ := \sigma(\beta-r, \beta]$ are disjoint. On the other hand, if I is c -flat, then we can color all backward arcs incident with I with at most $3c$ colors in such a way that each γ -cut of σ restricted to those arcs contains arcs of mutually different colors. This can be achieved e.g. by greedy coloring the γ -cuts for consecutive $\gamma = \alpha, \dots, \beta + 1$. Formally, there exists a function $\xi: \overline{A}_\sigma(T) \cap A(I, V) \rightarrow [3c]$ such that for every $\gamma \in [|V| - 1]$ and every two distinct arcs $a_1, a_2 \in \text{cut}_\sigma[\gamma] \cap A(I, V)$ we have $\xi(a_1) \neq \xi(a_2)$. In the following fix such a function.

► **Definition 10.** Let ϕ be a partial immersion in an interval I that is $2r$ -long and c -flat. For each $a \in A(H)$ we define the (r, c) -type $\tau^{(r,c)}(\phi(a))$ of the scattered path $\phi(a) = (P_i)_{i=1}^q$ as the following sequence of length $2q$:

$$(f_-(\text{first}(P_1)), f_+(\text{last}(P_1)), f_-(\text{first}(P_2)), f_+(\text{last}(P_2)), \dots, f_-(\text{first}(P_q)), f_+(\text{last}(P_q))),$$

where the functions $f_{\pm}: A(T[I]) \cup \partial(I) \cup \Gamma(I) \rightarrow [-3c] \cup [r] \cup \{X, H\}$ are defined as follows

$$f_-(a) = \begin{cases} -\xi(a) & \text{if } a \in \partial^-(I), \\ \sigma(\text{head}(a)) - \alpha & \text{if } a \in \Gamma^-(I) \text{ and } \text{head}(a) \in I_r^-, \\ X & \text{if } a \in \Gamma^-(I) \text{ and } \text{head}(a) \notin I_r^-, \\ H & \text{otherwise;} \end{cases}$$

$$f_+(a) = \begin{cases} -\xi(a) & \text{if } a \in \partial^+(I), \\ r + \sigma(\text{tail}(a)) - \beta & \text{if } a \in \Gamma^+(I) \text{ and } \text{tail}(a) \in I_r^+, \\ X & \text{if } a \in \Gamma^+(I) \text{ and } \text{tail}(a) \notin I_r^+, \\ H & \text{otherwise.} \end{cases}$$

Let S be the set of all terms of the sequences $(\tau^{(r,c)}(\phi(a)))_{a \in A(H)}$, where by term we mean a value with an assigned position (i.e. equal values in different sequences are considered different terms). The (r, c) -type of ϕ is the collection of types $\tau^{(r,c)}(\phi) = (\tau^{(r,c)}(\phi(a)))_{a \in A(H)}$ equipped with a pair of equivalence relations (R_-, R_+) on the set $S \cup [c]$ defined as follows.

- If a piece P_1 of $\phi(a_1)$ and a piece P_2 of $\phi(a_2)$ satisfy $\text{head}(\text{first}(P_1)) = \text{head}(\text{first}(P_2))$, then the corresponding terms in $\tau^{(r,c)}(\phi)$ (elements of the set $\{X\} \cup [r]$ with assigned positions in respective sequences) are in relation R_- .
- If a piece P of $\phi(a)$ is such that $\text{head}(\text{first}(P))$ is the tail of a singular arc of color $x \in [3c]$, then the corresponding term in $\tau^{(r,c)}(\phi)$ is in relation R_- with x .
- Analogously, if $\text{tail}(\text{last}(P_1)) = \text{tail}(\text{last}(P_2))$, then the terms $f_+(\text{last}(P_1))$ and $f_+(\text{last}(P_2))$ are in relation R_+ . And $x \in [3c]$ is in relation with all terms corresponding to tails of ends of paths which are simultaneously the head of the singular arc of color x .

Let us provide some intuition on what kind of information is stored in the type of ϕ defined above. First of all, the entire “singular interface” of this partial immersion is kept, i.e. in the type we remember precisely the singular arcs used to enter or exit I when traversing along each scattered path $\phi(a)$. Moreover, if we enter or exit I with a generic arc, we remember the precise vertex of entry/exit inside I , but only if it is close enough to the “border” of I (i.e. within the first or last r vertices in σ). Otherwise we remember the respective entry/exit as “generic”, which is marked by the marker X . Moreover, we keep the information about whether the scattered path begins or ends inside I – the marker H represents that a vertex of H is mapped under the partial immersion embedding to a vertex within I . Finally if the extreme (first or last) arcs of some pieces are generic and have the same first/last vertex in I , we remember this fact in the equivalence relations R_{\pm} . We shall need this information to be able to “glue” two partial immersions without using the same generic gluing arc for different pieces. The incidence with singular arcs is also stored to avoid a situation when one attempts a generic gluing along a singular arc. The reason for colors being stored as negative integers is purely technical – it ensures that $[r] \cap [-3c] = \emptyset$.

An (r, c) -type is any collection of sequences of even lengths over the set $[-3c] \cup [r] \cup \{X, H\}$ indexed by $A(H)$ and equipped with a pair of equivalence relations (R_-, R_+) on the union of the set of all terms of these sequences and $[3c]$. An I -admissible (r, c) -type is every type τ such that there exists a partial immersion ϕ in I such that $\tau = \tau^{(r,c)}(\phi)$. The size of a type is the sum of lengths of its sequences (i.e. it does not depend on the equivalence relations).

26:10 Polynomial Kernel for Immersion Hitting in Tournaments

Let γ_r be a function mapping each element from the set $[r]$ onto the single marker X , and identity otherwise. Intuitively, the function γ_r keeps only the information about generic nature of the end of a piece, and “forgets” about the closeness of this vertex to the boundary. We say that an (r, c) -type τ' is a *shortening* of an (r, c) -type τ if for every $a \in A(H)$ the sequence $\gamma(\tau'_a)$ is a subsequence of $\gamma(\tau_a)$ and the two sequences have the same first and last terms.

The following observation is a direct consequence of the definition of shortening of a partial immersion.

► **Observation 11.** *If a partial immersion ϕ' of (r, c) -type τ' is a shortening of a partial immersion ϕ of (r, c) -type τ , then τ' is a shortening of τ .*

► **Definition 12.** *An I -admissible (r, c) -type τ is called minimal (in I) if there is no I -admissible type τ' of strictly smaller size that would be a shortening of τ .*

► **Observation 13.** *Let ϕ be a partial immersion in I of (r, c) -type τ . Then ϕ is minimal if and only if τ is minimal in I .*

Proof. If ϕ is not minimal, then for any nontrivial shortening ϕ' of ϕ , $\tau^{(r,c)}(\phi')$ is a shortening of τ of strictly smaller size. Conversely, if τ is not minimal, then there exists an I -admissible type τ' of strictly smaller size. Every partial immersion ϕ' of type τ' is a nontrivial shortening of ϕ . ◀

We now observe that in a minimal partial immersion on an interval I , all scattered paths will be relatively small, that is, will visit I only a bounded number of times.

► **Lemma 14** (\times). *Suppose that $F \subseteq A(T)$ has the property that $|F| \leq f$, I is $4\|H\|(c+f+1)$ -long and c -flat, and a partial immersion ϕ in I is minimal and disjoint with F . Then for every $a \in A(H)$ the scattered path $\phi(a)$ has size at most $2c+3$.*

We call an (r, c) -type τ *short* if for every $a \in A(H)$ the length of $\tau(\phi(a))$ is at most $4c+6$.

► **Corollary 15** (\times). *For $r \geq 1$ we have the following:*

1. *If an interval I is $4\|H\|(r+c)$ -long and c -flat, and a partial immersion ϕ in I is minimal, then $\tau^{(r,c)}(\phi)$ is short.*
2. *For every pair of integers $r, c \in \mathbb{N}$ there is a constant $t(r, c)$ such that there are exactly $t(r, c)$ short (r, c) -types. In particular, for any interval I as above, there are at most $t(r, c)$ different I -admissible minimal (r, c) -types.*

Let I_1, I_2 be two consecutive c -flat, $2(r+c)$ -long σ -intervals and $I = I_1 \cup I_2$. We say that two (r, c) -types τ_1 of I_1 and τ_2 of I_2 are *compatible* if there exists a partial immersion ϕ in I such that $\phi|_{I_1}$ and $\phi|_{I_2}$ are compatible, $\phi|_{I_1}$ has type τ_1 , and $\phi|_{I_2}$ has type τ_2 .

The *gluing* of the types $\tau_1 \oplus \tau_2$ is defined as the set of all (r, c) -types of all such ϕ . The following lemma proves that this definition is correct, i.e. that types of two immersions store enough information to ensure the possibility of gluing them.

► **Lemma 16** (\times). *Let I_1, I_2 be two consecutive c -flat, $4\|H\|(r+c)$ -long σ -intervals and $I = I_1 \cup I_2$. If two (r, c) -types τ_1 of I_1 and τ_2 of I_2 are compatible, then for every partial immersion ϕ_1 in I_1 of type τ_1 and for every partial immersion ϕ_2 in I_2 of type τ_2 , the immersions ϕ_1 and ϕ_2 are compatible.*

Boundaried intervals and signatures. In the process of replacing protrusions we will need to consider intervals not as subsets of an ordered vertex set of a larger tournament, but as standalone structures which can be used to replace one another. We introduce the notion of an (r, c) -boundaried interval to enable such considerations.

► **Definition 17.** An (r, c) -boundaried interval is a digraph D on vertex set $V(D) = S^+ \cup I \cup S^-$ equipped with an ordering σ_I of I . Furthermore, we require the following:

- $|I| \geq 4\|H\|(r + c)$;
- $D[I]$ is a tournament;
- σ_I has width at most c ;
- $S^- \subseteq [3c] \times \{-\}$ and $S^+ \subseteq [3c] \times \{+\}$;
- each vertex of S^+ has only one incident arc and this arc belongs to the set $\vec{A}(I, S^+)$;
- each vertex of S^- has only one incident arc and this arc belongs to the set $\vec{A}(S^-, I)$.

The r -boundary of D is the pair of sets I^- and I^+ consisting of the first and the last r vertices of I in σ_I , respectively. For D defined above, we will shortly write $D = I \cup S^\pm$.

Note that this notion emulates a $4\|H\|(r + c)$ -long c -flat interval I in the following sense. Arcs whose heads are contained in S^+ correspond to $\partial^+(I)$, and arcs whose tails are contained in the set S^- correspond to $\partial^-(I)$. The names of the auxiliary vertices in S^\pm correspond to the ξ -colors of the respective backward arcs. The r -boundary consists of precisely those vertices whose generic entry or exit is remembered in the (r, c) -type of a partial immersion.

Formally, every $4\|H\|(r + c)$ -long c -flat σ -interval I in T can be uniquely encoded with an (r, c) -boundaried interval $D^{(r,c)}(I)$, whose structure resembles the structure of $T[I]$ and $\partial(I)$ as follows:

- the vertices and arcs of I are kept along with their ordering in T ;
- every singular arc $a \in \partial^\pm(I)$ is mapped to an arc joining $(\xi(a), \pm) \in S^\pm$ with the endpoint of a contained in I ;
- the projection of S^\pm onto the first coordinate is precisely $\{\xi(a) \mid a \in \partial^\pm(I)\}$.

The notion of a partial immersion can be naturally adjusted to the setting of boundaried intervals. The only difference is the lack of the “generic interface” i.e. there are no auxiliary edges in boundaries intervals used to emulate $\Gamma(I)$. These can be, however, emulated by storing the information from the type (marker X or number in $[r]$ if the generic arc is incident with the r -boundary, and the equivalence relations R_\pm) instead of the identity of particular generic arcs. Formally, a piece of a scattered path in (r, c) -boundaried interval can begin or end with an element in $\{X\} \cup [r]$ instead of a generic arc. In particular, this slightly modified variant of partial immersions can be equipped with precisely the same definition of admissible (r, c) -type as in the former case.

► **Definition 18.** An (r, c) -signature is a subset of the set of all short (r, c) -types. The (r, c) -signature $\Sigma^{(r,c)}(I)$ of a $4\|H\|(r + c)$ -long c -flat σ -interval I is the set of all I -admissible minimal (r, c) -types. The (r, c) -signature $\Sigma^{(r,c)}(D)$ of an (r, c) -boundaried interval $D = I \cup S^\pm$ is the set of all I -admissible minimal (r, c) -types.

The intuition behind this definition is that if I is appropriately long and flat, then the signature of I stores the information about all possible interactions of I with minimal partial immersions. Note that $\Sigma^{(r,c)}(D^{(r,c)}(I)) = \Sigma^{(r,c)}(I)$.

We say that two (r, c) -boundaried intervals $I \cup S^\pm$ and $I' \cup S'^\pm$ are *exchangeable* if they have equal (r, c) -signatures, $S^\pm = S'^\pm$ (both intervals use precisely the same colors on the boundary), and the incidence structure of r -boundaries of those intervals with backward arcs

26:12 Polynomial Kernel for Immersion Hitting in Tournaments

is the same, i.e. for every $i \in [r]$ the set of colors of singular arcs incident with both S^\mp and the i -th vertex of I^\pm is the same as analogously defined set of colors for i -th vertex of I'^\pm . Intuitively this means that in T we may replace the interval I with I' .

► **Corollary 19.** *For every pair of integers $r, c \in \mathbb{N}$ there is a constant $s(r, c)$ such that there are exactly $s(r, c)$ different (r, c) -signatures.*

Proof. We may set $s(r, c) = 2^{t(r, c)}$, where $t(r, c)$ is the constant provided by Corollary 15. ◀

For future discussion of algorithmic aspects, we will need the following observation.

► **Lemma 20** (\asymp). *Consider r and c fixed and let T , σ , and I be as in Definition 18. Then given T , σ , and I , the signature $\Sigma^{(r, c)}(I)$ can be computed in polynomial time.*

Let $\mathcal{S}^{(r, c)}$ be the set of all (r, c) -signatures; we have $|\mathcal{S}^{(r, c)}| = s(r, c)$, where $s(r, c)$ is the constant given by Corollary 19. Let $\mathcal{S}_\sigma^{(r, c)}$ be the set of those (r, c) -signatures which are equal to $\Sigma^{(r, c)}(I)$ for some σ -interval I , i.e. contain only I -admissible minimal (r, c) -types. Then $\mathcal{S}_\sigma^{(r, c)} \subseteq \mathcal{S}^{(r, c)}$, so $|\mathcal{S}_\sigma^{(r, c)}| \leq s(r, c)$. We argue that $\mathcal{S}_\sigma^{r, c}$ has a structure of a semigroup in the following sense.

► **Lemma 21** (\asymp). *Let I_1, I_2 be two $4\|H\|(r+c)$ -long c -flat σ -intervals such that $I = I_1 \cup I_2$ is a c -flat σ -interval. Then $\Sigma^{(r, c)}(I)$ is uniquely determined by $\Sigma^{(r, c)}(I_1)$ and $\Sigma^{(r, c)}(I_2)$.*

Lemma 21 implies that the set $\mathcal{S}_\sigma^{(r, c)} \cup \{0\}$ can be endowed with an associative binary product operation such that for every two consecutive intervals I_1, I_2 , the product of their signatures is the signature of their union $I_1 \cup I_2$. Formally, we set the product for all pairs of consecutive intervals as above; Lemma 21 ensures that this is well-defined. Next, for all pairs of elements $\tau_1, \tau_2 \in \mathcal{S}_\sigma^{r, c}$ for which their product is not yet defined, we set $\tau_1 \cdot \tau_2 = 0$. Also, we set $0 = 0 \cdot 0 = 0 \cdot \tau = \tau \cdot 0$ for all $\tau \in \mathcal{S}_\sigma^{(r, c)}$. In this way, $\mathcal{S}_\sigma^{(r, c)} \cup \{0\}$ becomes a monoid; the empty signature is the neutral element of multiplication.

By Lemma 5 we obtain the following.

► **Corollary 22** (\asymp). *Suppose I is a c -flat $4\|H\|(r+c)\ell^{3s(r, c)}$ -long σ -interval. Then there exists a sequence of consecutive $4\|H\|(r+c)$ -long c -flat σ -intervals $(I_i)_{i=1}^\ell$ whose (r, c) -signatures are equal and equal to the signature of their union. Moreover, given r, c, T, σ , and I , such a sequence can be found in polynomial time.*

4 Finding protrusions

In order to find an appropriately large subgraph of T which does not “affect” the behavior of T with respect to H -HITTING IMMERSIONS IN TOURNAMENTS, we roughly proceed as follows. First, we find a suitable ordering σ of $V(T)$ and an appropriately long interval X in σ such that X has a constant-size singular interface towards the remainder of T . Then, inside X , we find (again, an appropriately long) subinterval I of a very specific structure: I can be divided into $2k + 3$ subintervals with the same signatures as itself. This is where we use Simon Factorization through Corollary 22. In the next part we use this extra structure to prove that one of these subintervals can be replaced with a strictly smaller replacement in such a way that after the substitution, we obtain an equivalent instance of the problem.

We proceed to a formal implementation of this plan. The first lemma gives the ordering σ and the interval X .

► **Lemma 23** (\asymp). *Let T be a tournament with $\text{ctw}(T) \leq c$ and $|T| \geq (2c+1)(x+1)(k+1)$. If T contains at most k arc-disjoint immersion copies of H , then there exists an ordering σ of $V(T)$ and an H -free σ -interval X such that $|X| \geq x$ and X is c_H -flat with respect to σ .*

Moreover, given T, k, c, x as above, one can in polynomial time either conclude that T contains more than k arc-disjoint immersion copies of H , or find an ordering σ and an interval X satisfying the above properties.

In the remainder of this section let $x \geq 4\|H\|$, r, c be fixed positive integers. Moreover, let T be a tournament for which there exists an optimal solution of size at most k and let σ be an ordering of T . Finally let X be an H -free c -flat $x(3c+k+1)(2k+3)^{3s(r,c)}$ -long σ -interval. We assume that there is a coloring ξ mapping all σ -backward arcs incident to X to colors in $[3c]$ such that not two such arcs of the same color participate in the same σ -cut.

We may apply Corollary 22 to X to find a collection of $2k+3$ consecutive σ -intervals I_i with $|I_i| = x(3c+k+1)$ for all $i \in [2k+3]$ such that all I_i have equal (r, c) -signatures, and this common signature, call it Σ , is equal to the (r, c) -signature of their union I . Then I is an H -free c -flat $x(3c+k+1)(2k+3)$ -long σ -interval. That such I can be found in polynomial time (given r, c, x, T, X, σ , and ξ) follows from Corollary 22.

Now comes a key step in the proof: we argue that from the equality of types of I, I_1, \dots, I_{2k+3} it follows that every optimum solution will contain a bounded number of arcs incident with I .

► **Lemma 24** (\asymp). *For every optimal solution $F \subseteq A(T)$, we have $|F \cap A(I, V(T))| \leq 2c$.*

We define a digraph T° based on T and $I = I_1 \cup \dots \cup I_{2k+3}$ as follows: start with T , and

- remove all vertices of I_2 ;
- for every arc $a \in \partial^+(I_2)$, replace a with an arc with the same head as a and tail in a fresh vertex $s_{\xi(a)}^+$;
- for every arc $a \in \partial^-(I_2)$, replace a with an arc with the same tail as a and head in a fresh vertex $s_{\xi(a)}^-$.

We call the constructed graph a *c -boundaried co-interval*. The intuition of this construction is as follows. We pinch off one of the $2k+3$ intervals and keep the singular arc interface in a fashion similar as in (r, c) -boundaried intervals. The only difference is that we do not keep track of the r -boundary vertices.

Now we can define the *gluing* of T° with an (r, c) -boundaried interval $B = I \cup S^\pm$ with signature Σ , simply by identifying the singular arcs of the same color (note that different vertices from S^\pm can be therefore mapped to the same vertex) and completing the obtained structure to a tournament by making all missing arcs generic. Note that in order for this to be well-defined, we need to require that the sets of colors of the singular arcs in T° and in B are identical – if it is so, we will say that T° and B are *compatible*. Also, note that we require that the signature of B is Σ : that is, the possible types of partial immersions present in B are exactly the same as in the substituted interval I_2 .

Denote by $T^\circ \oplus B$ the tournament obtained from gluing T° and B . Note that in this tournament we have naturally defined ordering of vertices: in T° it is inherited from the ordering σ of T and within B it is inherited from the ordering σ_I of the boundaried interval. Finally all the vertices of the substituted interval appear in the ordering between the two interval parts (prefix and suffix) of the co-interval. The following observation is obvious.

► **Observation 25.** *If two exchangable (r, c) -boundaried intervals $B = I \cup S^\pm$ and $B' = I' \cup S^\pm$ are compatible with T° , then the (r, c) -signatures of I in $T^\circ \oplus B$ and I' in $T^\circ \oplus B'$ are equal.*

26:14 Polynomial Kernel for Immersion Hitting in Tournaments

We now observe, by inspecting the proof of Lemma 24, that if in I we replace one of its subintervals with an exchangeable interval, then the conclusion of Lemma 24 – that the modified I' will still have constant incidence with every optimal solution in T – will still hold. Let us summarize this section with putting together these observations and recalling all needed assumptions.

► **Corollary 26** (\asymp). *Suppose that $x \geq 4\|H\|$, r, c are fixed positive integers, T is a tournament of for which there exists an optimal solution of size at most k and σ is an ordering of T . Suppose further that X is an H -free c -flat $x(3c + k + 1)(2k + 3)^{3s(r,c)}$ -long σ -interval with all incident σ -backward arcs colored according to ξ with colors $[3c]$ so that no two arcs of the same color participate in the same σ -cut.*

Then there exists an (r, c) -signature Σ , a c -boundaried co-interval T° of this signature, and an (r, c) -boundaried interval $B = I \cup S^\pm$ such that $T = T^\circ \oplus B$ and moreover for every $B' = I' \cup S'^\pm$ exchangeable with B and for every optimal solution $F \subseteq A(T')$ where $T' = T^\circ \oplus B'$ we have: $|F \cap A(I', T')| \leq 2c$. Moreover, given x, r, c, T, σ , and X , such Σ, T° , and B can be computed in polynomial time.

5 Replacing protrusions

In the entire section we fix $c := c_H$, where c_H is the constant from Corollary 4, and $r := 6\|H\|c$. Moreover we assume that $x \geq 4\|H\|$; the precise value of x will be determined later.

► **Definition 27.** *Protrusion is any (r, c) -boundaried interval $X = I \cup S^\pm$ which is H -free and $x(3c + k + 1)$ -long. For brevity we will refer to X as to I . The set $\Sigma^{(r,c)}(X)$ is the signature of the protrusion.*

Let T be a tournament equipped with a vertex ordering σ and let I be an H -free interval and such that $T = T^\circ \oplus X$, where X is a protrusion of signature Σ . Let $I \subseteq V(T)$ be the interval defined by the protrusion. Fix the coloring ξ of σ -backward arcs incident with I .

Recall that from Corollary 26 follows that for every (r, c) -boundaried co-interval T° compatible with X if $T^\circ \oplus X$ admits an optimal solution F of size not greater than k , then there are at most $2c$ arcs in F incident with X .

For every protrusion $X = I \cup S^\pm$ define a function $f_X: 2^{\mathcal{S}^{(r,c)}} \rightarrow \{0, 1, 2, \dots, 2c\} \cup \{\infty\}$ as follows: $f_X(S)$ is the minimum number of arcs in $A(I, I)$ needed to hit all partial immersions in I whose signatures belong to S , or ∞ if this number is greater than $2c$.

We introduce an equivalence relation \sim on the set of all protrusions. Let $X = I \cup S^\pm$, $X' = I' \cup S'^\pm$. We say that $X \sim X'$ if:

- $S^\pm = S'^\pm$;
- $\Sigma^{(r,c)}(X) = \Sigma^{(r,c)}(X')$; and
- $f_X(S) = f_{X'}(S)$ for every $S \subseteq \mathcal{S}^{(r,c)}$.

Note that the number of equivalence classes of \sim is finite and bounded by a constant depending only on H . This means that if in each class we pick a representative with the minimal number of vertices (call each such element a *small* protrusion), then all small protrusions will have size bounded from above uniformly by a constant s_H depending on the digraph H only.

Same arguments as in the proof of Lemma 20 give the following.

► **Lemma 28.** *Given protrusions X and X' it can be decided in polynomial time whether $X \sim X'$.*

We now argue that equivalent protrusions are replaceable.

► **Lemma 29** (\asymp). *Suppose that $T = T^\circ \oplus X$ is a tournament satisfying the conclusion of Corollary 26, where X is a boundaried (r, c) -interval of signature Σ and T° is a c -boundaried co-interval which is compatible with X .*

Then for every X' such that $X \sim X'$, the optimal solution in $T^\circ \oplus X$ is of size not greater than k if and only if the optimal solution in $T^\circ \oplus X'$ is of size not greater than k .

We are now set up with all tools needed to prove our main theorem.

Proof of Theorem 1. We prove that provided $|T| > C \cdot k^C$, where the constant C will be defined later, one can compute an instance (T', k) equivalent to (T, k) and satisfying $|T'| < |T|$. The conclusion will follow from applying such reduction (at most) $|T|$ times.

First of all note that from Theorem 3 for a digraph being a disjoint union of k exemplars of H , we conclude that if T does not contain k arc-disjoint immersion copies of H , then $\text{ctw}(T) \leq c_0 k^2$ for some constant c_0 (depending on H). In particular, it follows that if $\text{ctw}(T) > k^2 c_0$ (which can be established in polynomial time using Lemma 2), then T is a NO-instance. So from now on we may assume that $\text{ctw}(T) \leq c_0 k^2$.

Let $c = c_H$, $r = 6\|H\|c$, $x' = \max\{4\|H\|, s_H + 1\}$ and $x = x'(3c + k + 1)(2k + 3)^{3s(r,c)}$, where $s(r, c)$ is defined in Corollary 19. Let C be a constant satisfying $Ck^C \geq (2k^2 c_0 + 1)(x + 1)(k + 1)$, e.g. $C = \max\{3s(r, c) + 4, 5^{3s(r,c)} \cdot 3c_0 \cdot 4x' \cdot (6c + 2)\}$.

Suppose that T is a tournament satisfying $\text{ctw}(T) \leq k^2 c_0$ and $|T| > Ck^C$. Applying Lemma 23, we either conclude that T admits more than k arc-disjoint copies of H (so (T, k) is a NO-instance), or find an ordering σ of $V(T)$ and an H -free c -flat x -long σ -interval J . Both conclusions can be effectively gained in polynomial time.

In the latter case, we may use Corollary 22 in a manner described in Section 4 to find in J an H -free c -flat $x'(3c + k + 1)(2k + 3)$ -long σ -interval I of (r, c) -signature Σ , which can be decomposed to $2k + 3$ consecutive $x'(3c + k + 1)$ -long σ -intervals I_i , each of (r, c) -signature Σ . Both I and Σ are found in polynomial time.

Let $T = T^\circ \oplus X$ be the decomposition where T° is a c -boundaried co-interval and $X = (I_2 \cup S^\pm, \Sigma)$ is a protrusion corresponding to I_2 . Clearly, X is compatible with T° . Using Lemma 28 we may check in polynomial time all small protrusions to find one X' such that $X' \sim X$. Let us define $T' = T^\circ \oplus X'$.

By Lemma 29 we conclude that (T', k) is an instance of H -HITTING IMMERSIONS IN TOURNAMENTS equivalent to (T, k) . Moreover as $|X'| \leq s_H < |X|$, we have that $|T'| < |T|$. ◀

References

- 1 Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In *36th International Colloquium on Automata, Languages and Programming, ICALP 2009*, volume 5555 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2009. doi:10.1007/978-3-642-02927-1_6.
- 2 Florian Barbero, Christophe Paul, and Michał Pilipczuk. Exploring the complexity of layout parameters in tournaments and semicomplete digraphs. *ACM Trans. Algorithms*, 14(3):38:1–38:31, 2018. doi:10.1145/3196276.
- 3 Stéphane Bessy, Marin Bougeret, R. Krithika, Abhishek Sahu, Saket Saurabh, Jocelyn Thiebaut, and Meirav Zehavi. Packing arc-disjoint cycles in tournaments. *Algorithmica*, 83(5):1393–1420, 2021. doi:10.1007/s00453-020-00788-2.
- 4 Stéphane Bessy, Fedor V. Fomin, Serge Gaspers, Christophe Paul, Anthony Perez, Saket Saurabh, and Stéphan Thomassé. Kernels for feedback arc set in tournaments. *Journal of Computer and System Sciences*, 77(6):1071–1078, 2011. doi:10.1016/j.jcss.2010.10.001.



- 5 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. doi:10.1145/2973749.
- 6 Mikolaj Bojanczyk. Factorization forests. In *13th International Conference on Developments in Language Theory, DLT 2009*, volume 5583 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2009. doi:10.1007/978-3-642-02737-6_1.
- 7 Marthe Bonamy and Michał Pilipczuk. Graphs of bounded cliquewidth are polynomially χ -bounded. *Advances in Combinatorics*, 2020:8, 2020.
- 8 Łukasz Bożyk and Michał Pilipczuk. On the Erdős-Pósa property for immersions and topological minors in tournaments. *Discrete Mathematics & Theoretical Computer Science*, vol. 24, no. 1, April 2022. doi:10.46298/dmtcs.7099.
- 9 Maria Chudnovsky, Alexandra Ovetsky Fradkin, and Paul D. Seymour. Tournament immersion and cutwidth. *Journal of Combinatorial Theory, Series B*, 102(1):93–101, 2012. doi:10.1016/j.jctb.2011.05.001.
- 10 Maria Chudnovsky and Paul D. Seymour. A well-quasi-order for tournaments. *Journal of Combinatorial Theory, Series B*, 101(1):47–53, 2011. doi:10.1016/j.jctb.2010.10.003.
- 11 Uriel Feige. Faster FAST (Feedback Arc Set in Tournaments). *CoRR*, abs/0911.5094, 2009. arXiv:0911.5094.
- 12 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*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.
- 13 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.
- 14 Fedor V. Fomin and Michał Pilipczuk. On width measures and topological problems on semi-complete digraphs. *Journal of Combinatorial Theory, Series B*, 138:78–165, 2019. doi:10.1016/j.jctb.2019.01.006.
- 15 Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *ACM Transactions on Algorithms*, 13(3):35:1–35:35, 2017. doi:10.1145/3029051.
- 16 Archontia C. Giannopoulou, Ken-ichi Kawarabayashi, Stephan Kreutzer, and O-jeung Kwon. The directed flat wall theorem. In *2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 239–258. SIAM, 2020. doi:10.1137/1.9781611975994.15.
- 17 Archontia C. Giannopoulou, Ken-ichi Kawarabayashi, Stephan Kreutzer, and O-jeung Kwon. Directed tangle tree-decompositions and applications. In *2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 377–405. SIAM, 2022. doi:10.1137/1.9781611977073.19.
- 18 Archontia C. Giannopoulou, Michał Pilipczuk, Jean-Florent Raymond, Dimitrios M. Thilikos, and Marcin Wrochna. Linear kernels for edge deletion problems to immersion-closed graph classes. *SIAM Journal on Discrete Mathematics*, 35(1):105–151, 2021. doi:10.1137/18M1228839.
- 19 Marek Karpinski and Warren Schudy. Faster algorithms for Feedback Arc Set Tournament, Kemeny Rank Aggregation and Betweenness Tournament. In *21st International Symposium on Algorithms and Computation, ISAAC 2010*, volume 6506 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2010. doi:10.1007/978-3-642-17517-6_3.
- 20 Ken-ichi Kawarabayashi and Stephan Kreutzer. The directed grid theorem. In *47th Annual ACM Symposium on Theory of Computing, STOC 2015*, pages 655–664. ACM, 2015. doi:10.1145/2746539.2746586.
- 21 Ilhee Kim and Paul D. Seymour. Tournament minors. *Journal of Combinatorial Theory, Series B*, 112:138–153, 2015. doi:10.1016/j.jctb.2014.12.005.
- 22 Manfred Kufleitner. The height of factorization forests. In *33rd International Symposium Mathematical Foundations of Computer Science 2008, MFCS 2008*, volume 5162 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 2008. doi:10.1007/978-3-540-85238-4_36.

- 23 Jaroslav Nešetřil, Patrice Ossona de Mendez, Michał Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Rankwidth meets stability. In *2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 2014–2033. SIAM, 2021. doi:10.1137/1.9781611976465.120.
- 24 Jaroslav Nešetřil, Roman Rabinovich, Patrice Ossona de Mendez, and Sebastian Siebertz. Linear rankwidth meets stability. In *2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 1180–1199. SIAM, 2020. doi:10.1137/1.9781611975994.72.
- 25 Alexandra Ovetsky Fradkin. *Forbidden structures and algorithms in graphs and digraphs*. PhD thesis, Princeton University, 2011.
- 26 Alexandra Ovetsky Fradkin and Paul D. Seymour. Tournament pathwidth and topological containment. *Journal of Combinatorial Theory, Series B*, 103(3):374–384, 2013. doi:10.1016/j.jctb.2013.03.001.
- 27 Alexandra Ovetsky Fradkin and Paul D. Seymour. Edge-disjoint paths in digraphs with bounded independence number. *Journal of Combinatorial Theory, Series B*, 110:19–46, 2015. doi:10.1016/j.jctb.2014.07.002.
- 28 Jean-Florent Raymond. Hitting minors, subdivisions, and immersions in tournaments. *Discrete Mathematics and Theoretical Computer Science*, 20(1), 2018. URL: <http://dmtcs.episciences.org/4212>.
- 29 Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72(1):65–94, 1990. doi:10.1016/0304-3975(90)90047-L.

A Systematic Study of Isomorphism Invariants of Finite Groups via the Weisfeiler-Leman Dimension

Jendrik Brachter  

TU Darmstadt, Germany

Pascal Schweitzer  

TU Darmstadt, Germany

Abstract

We investigate the relationship between various isomorphism invariants for finite groups. Specifically, we use the Weisfeiler-Leman dimension (WL) to characterize, compare and quantify the effectiveness and complexity of invariants for group isomorphism.

It turns out that a surprising number of invariants and characteristic subgroups that are classic to group theory can be detected and identified by a low dimensional Weisfeiler-Leman algorithm. These include the center, the inner automorphism group, the commutator subgroup and the derived series, the abelian radical, the solvable radical, the Fitting group and π -radicals. A low dimensional WL-algorithm additionally determines the isomorphism type of the socle as well as the factors in the derived series and the upper and lower central series.

We also analyze the behavior of the WL-algorithm for group extensions and prove that a low dimensional WL-algorithm determines the isomorphism types of the composition factors of a group.

Finally we develop a new tool to define a canonical maximal central decomposition for groups. This allows us to show that the Weisfeiler-Leman dimension of a group is at most one larger than the dimensions of its direct indecomposable factors. In other words the Weisfeiler-Leman dimension increases by at most 1 when taking direct products.

2012 ACM Subject Classification Theory of computation \rightarrow Complexity theory and logic; Mathematics of computing \rightarrow Combinatorial algorithms

Keywords and phrases group isomorphism problem, Weisfeiler-Leman algorithms, group invariants, direct product decompositions

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.27

Related Version *Full Version:* <https://arxiv.org/abs/2111.11908>

Funding The research leading to these results has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (EngageS: grant agreement No. 820148) and from the German Research Foundation DFG (SFB-TRR 195 “Symbolic Tools in Mathematics and their Application”).

1 Introduction

Tasks of classifying finite groups up to isomorphism and generating particular classes of finite groups are fundamental and recurring themes in computational group theory. Yet, in particular the computational complexity of such problems remains most illusive to date.

For example, for most orders up to 20.000 the number of non-isomorphic finite groups has been computed and the groups have been exhaustively generated [13]. But there are currently 38 notoriously difficult, exceptional cases, for which this information is beyond our current means (see [13]). The varying difficulty across different orders is in part caused by the erratic fluctuation of the number of isomorphism classes of finite groups as the order increases. This number appears to be closely linked to the multiplicities of the prime factors of the respective order, but even estimating the number of groups of a given order is non-trivial.



© Jendrik Brachter and Pascal Schweitzer;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 27; pp. 27:1–27:14
Leibniz International Proceedings in Informatics



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

Generation tasks for classes of groups have a long tradition dating back to Cayley [7]. Nowadays, there is extensive work on generating particular classes of groups. For example there are practically efficient algorithms for the generation of finite nilpotent or finite solvable groups [12]. However, the algorithms come without efficient running time guarantees.

One of the difficulties for a complexity analysis stems from the group isomorphism problem. Indeed, the group isomorphism problem for finite groups stays among the few standard tasks in computational group theory with uncertain complexity. In principle, we desire algorithms with an efficient worst case running time measured in the number of generators through which the groups are given. However, we do not even have algorithms with an efficient worst case running time when measured in the order of the group. In fact the only improvement for the worst case complexity over Tarjan's classic $n^{\log(n)+O(1)}$ algorithm are $n^{\frac{1}{c} \cdot \log(n)+O(1)}$ algorithms with a small constant c depending on the model of computation (randomization, quantum computing etc.) [21, 23, 24]. There is however a nearly-linear time algorithm that solves group isomorphism for most orders [11].

A closely related problem is that of computing isomorphism invariants to distinguish groups. Efficiently computable complete invariants are sufficient for general isomorphism testing. However, we do not know efficiently computable complete invariants even for very special cases, such as nilpotent p -groups of class 2. Partial invariants only give incomplete isomorphism tests, but they still find application in generation tasks allowing for heuristic fast pruning [13]. Given the long history of (algorithmic) group theory, there is an abundance of partial invariants.

Generally the techniques involved in generation and isomorphism computations exploit the existence of various characteristic subgroups classic to group theory. As outlined in [13], these include exploiting the Frattini subgroup $\Phi(G)$ [3], the exponent- p -central series [22], characteristic series [25] and similar.

Overall, many of the techniques currently in use are ad-hoc, focused on practical performance, and do not lead to efficient worst case upper bounds for the complexity of the algorithmic problems. As a consequence, the general picture for finite groups is somewhat chaotic. There is often no structured way of comparing or combining invariants for group isomorphism. E.g., two given invariants may be incomparable in their distinguishing power, making it unclear which invariant to use. Also the required time to evaluate an invariant may be difficult to estimate and can depend significantly on the input group. Even when we are given a class of efficiently computable invariants, it will generally be unclear which invariants to choose or how to efficiently combine their evaluation algorithmically.

In Summary, we lack the formal means to characterize, compare, or quantify the effectiveness and complexity of invariants for group isomorphism. We therefore propose a systematic study of computationally tractable invariants for finite groups.

For inspiration on how to systematize such a study, we turn to algorithmic finite model theory and specifically descriptive complexity theory. This allows us to characterize the complexity of an invariant by considering a formula within a logic that captures the invariant. A natural choice for a logic from which to choose the formulas is the powerful fixed point logic with counting. Not only can this logic express all polynomial time computable languages on ordered structures [17, 26], but in the context of graphs it has also proven to be an effective tool in comparing invariants (see [20]). As a measure for the complexity of an invariant we can then use the number of variables required to express the invariant in fixed point logic with counting. Crucially there is a corresponding algorithm, the k -dimensional Weisfeiler-Leman algorithm (WL-refinement, WL), that (implicitly) simultaneously evaluates all invariants that are expressible by formulas requiring at most $k + 1$ variables in polynomial time¹.

¹ For groups there are actually two natural, closely related versions of the logic and of the algorithm,

Thus, to enable a quantification and comparison of the complexity of invariants we suggest the Weisfeiler-Leman algorithm. More specifically we suggest to use the Weisfeiler-Leman dimension, which determines how many variables are required to express a given invariant as a formula. This gives us a natural and robust framework for studying group invariants. In fact, the k -dimensional Weisfeiler-Leman algorithm is universal for all invariants of the corresponding dimension, resolving the issue of how to combine invariants. With this approach we also include an abundance of invariants that have not been considered before. However, it is a priori not clear at all that commonly used invariants can even be captured by the framework, i.e., that they even have bounded WL-dimension.

Contribution. The first contribution of this paper is to show that a surprising number of isomorphism invariants and subgroups that are classic to group theory can be detected and identified by a low dimensional Weisfeiler-Leman algorithm.

Specifically, we show first that for a small value of k , groups not distinguished by k -WL_{II} have centers ($k \geq 2$), inner automorphism groups ($k \geq 4$), derived series ($k \geq 3$), abelian radicals ($k \geq 3$), solvable radicals ($k \geq 2$), Fitting groups ($k \geq 3$) and π -radicals ($k \geq 3$) that are indistinguishable by k -WL_{II}. They also have isomorphic socles ($k \geq 5$), stepwise isomorphic factors in the derived series ($k \geq 4$), upper central series ($k \geq 4$), and lower central series ($k \geq 4$). Our techniques regarding characteristic subgroups are fairly general. We thus expect them to be applicable to a large variety of other isomorphism invariants. In particular they should facilitate the analysis of combinations of invariants one might be interested in (such as the Fitting series or the hypercenter).

Beyond these characteristic subgroups, in our second contribution we show that composition factors are implicitly computed by a Weisfeiler-Leman algorithm of bounded dimension.

► **Theorem 1.1.** *If $k \geq 5$ and G is indistinguishable from H via k -WL_I, then G and H have the same (isomorphism types of) composition factors (with multiplicities).*

The theorem shows that the WL-algorithm, which is a purely combinatorial algorithm, can compute group theoretic invariants that do not even appear as a canonical subset of the group. In particular, the composition factors cannot be localized within the group, and at first sight it might not be clear that WL grasps quotient groups.

Our third contribution, having the most technical proof and building on our other results, regards direct products of groups. Here we consider the decomposition of a group into direct factors. We show that direct products indistinguishable by k -WL must arise from factors that are indistinguishable by $(k + 1)$ -WL.

► **Theorem 1.2.** *Let $G = G_1 \times \cdots \times G_d$ be a direct product and $k \geq 5$. If G and H are not distinguished by k -WL_{II} then there are direct factors $H_i \leq H$ such that $H = H_1 \times \cdots \times H_d$ and such that for all i the groups G_i and H_i are not distinguished by $(k - 1)$ -WL_{II}.*

In other words, the Weisfeiler-Leman dimension increases by at most 1 when taking direct products. The main difficulty here is that decompositions into direct products are not unique, and thus not definable. These complications arise mainly due to central elements. However we manage to define a canonical maximal central decomposition, that is generally finer than a decomposition into direct factors. We then show that this canonical decomposition is implicitly computed by the WL-algorithm.

k -WL_I and k -WL_{II}, see Section 3.

One way of interpreting our results is that the Weisfeiler-Leman algorithm comprises a unified way of computing all the mentioned invariants and characteristics simultaneously. The dimension can therefore be used to compare the complexity of invariants.

Techniques. To show the various results on characteristic subgroups, we prove a general result on group expressions. It essentially shows that subsets that can be defined by equation systems can be detected by k -WL (see Lemma 4.3).

The result on composition factors involves a technique that relates k -WL distinguishability of groups to detectable normal subgroups and detectable quotients (Theorem 4.8).

To deal with direct products, we extend the technique to simultaneously relate chains of subgroups in two indistinguishable groups (Lemma 4.9). Here we exploit well-known connections of pebble games to Weisfeiler-Leman algorithms. However, the main difficulty regarding our result on decompositions into direct factors is that such decompositions are not unique. In fact in general, a group element cannot be assigned to a direct factor in a well defined sense, making it impossible for WL to detect direct factors. For this purpose we develop a new technical tool, component-wise filtrations (Definition 6.8), which compensate for the non-uniqueness to extract at least the isomorphism type of the direct factors (Lemma 6.10). We also exploit the non-commuting graph of the group and show that certain subsets, which we call non-abelian components, can be detected by k -WL (Lemma 6.14). These non-abelian components lead to a WL-definable maximal central decomposition of every finite group.

Outline. Section 2 provides preliminaries. Section 3 treats WL-refinement in the context of colored groups. In Section 4, we show that invariants generated via WL-refinement fulfill group theoretic closure properties. Section 5 is an extensive collection of specific structure properties and invariants which Weisfeiler-Leman algorithms detect in finite groups. Finally, in Section 6 we investigate the ability of WL-refinement to detect direct product decompositions, building on the results of the previous sections. Throughout the paper various lemmas have been condensed and proofs are omitted. Attached is a full version of the paper (sections agree but the sections are expanded and numbers may disagree.).

Further related work. We should point out that there are various results in the literature on decomposing groups into indecomposable direct factors for various input models of groups. For example there is a polynomial time algorithm to decompose permutation groups into direct products [28]. Finally, there is a recent algorithm that finds direct product decompositions of permutation groups with factors having disjoint support [8]. There is also a polynomial time algorithm that computes direct factors efficiently for groups given by multiplication table [19]. Aspects of this algorithm are related to arguments we use for studying the behavior of WL on direct products (see the beginning of Section 6 for a discussion).

Regarding group isomorphism problems, for isomorphism of Abelian groups a linear time algorithm is known [18] and there are near linear time algorithms for some classes of non-abelian groups (e.g, [10]). Recent directions relate group isomorphism to tensor problems [15]. The Weisfeiler-Leman algorithm has also been incorporated as a subroutine within other sophisticated group isomorphism algorithms [6].

Regarding Weisfeiler-Leman algorithms, the literature is somewhat limited when it comes to groups [4, 6] but quite extensive when it comes to graphs. In [2], for example the authors investigate some graph invariants that are captured by the Weisfeiler-Leman algorithm. We refer to [20] for an introduction and extensive overview over recent results for WL on graphs.

2 Preliminaries

Sets & Partitions. We denote multisets as $\{\{\dots\}\}$. Given disjoint sets M and N , their union is $M \uplus N$. The m -th Cartesian power of M is $M^{(m)}$.

Graphs. We use $V(\Gamma)$ and $E(\Gamma)$ to refer to the vertices or edges of a graph Γ . For a subset $S \subseteq V(\Gamma)$, let $\Gamma[S]$ denote the subgraph induced by the set S .

Groups. Groups are assumed to be finite. The symmetric group on m symbols is denoted by S_m . The order of a group element $g \in G$ is the order of the group generated by g , i.e., $|g| := |\langle g \rangle|$. Given a finite set of primes π , a π -group is a group whose order is only divisible by primes in π . A group element is called a π -element if it generates a π -group. For any $d \in \mathbb{Z}$, set $(G)^d := \langle \{g^d \mid g \in G\} \rangle$ in contrast to the d -fold direct power G^d . Given $g, h \in G$, we define the commutator $[g, h] := ghg^{-1}h^{-1}$. Conjugation is denoted by $g^h := hgh^{-1}$. If $M, N \subseteq G$ we set $[M, N] := \langle [m, n] \mid m \in M, n \in N \rangle$ and we write $G' := [G, G]$.

3 Colored Groups & Weisfeiler-Leman Algorithms

We recapitulate various notions regarding WL-algorithms on groups. For WL on graphs we refer to [20]. For uncolored groups, versions of WL were defined in [4]. For our purpose, we need to generalize the concepts to the setting of colored groups. Let us point out that in the case of graphs, colors can be replaced by gadget constructions to obtain uncolored graphs while maintaining the graph's combinatorial properties. However, for groups it is unclear how to do this. Nevertheless, we can still use colors to restrict the set of possible automorphisms.

3.1 Colorings on Finite Groups

Given a natural number k and a finite group G , a (k) -coloring (over G) is just a map $\gamma : G^{(k)} \rightarrow \mathcal{C}$ where \mathcal{C} denotes some finite set of colors. A k -coloring γ partitions $G^{(k)}$ into color classes. We refer to 1-colorings as *element-colorings*.

The color set \mathcal{C} is often omitted. Considering two natural numbers $m < k$, a k -coloring $\gamma : G^{(k)} \rightarrow \mathcal{C}$ induces an m -coloring $\gamma^{(m)} : G^{(m)} \rightarrow \mathcal{C}$ via $\gamma^{(m)}((g_1, \dots, g_m)) := \gamma((g_1, \dots, g_m, 1, \dots, 1))$. To simplify notation, we may write γ again instead of $\gamma^{(m)}$ and instead of $\gamma^{(1)}$ we use $\gamma^{(G)}$ to emphasize that the coloring is pulled back to group elements.

► **Definition 3.1.** A colored group is a group G together with an element-coloring γ over G . We say $M \subseteq G$ is γ -induced if $\gamma(M) \cap \gamma(G \setminus M) = \emptyset$ holds, i.e., M is a union of γ -color classes. Colored groups (G, γ_G) and (H, γ_H) are isomorphic if there is a group isomorphism $\varphi : G \rightarrow H$ that respects colors, i.e., $\gamma_H \circ \varphi = \gamma_G$. We set $\text{Aut}_{\gamma_G}(G) := \{\varphi \in \text{Aut}(G) \mid \gamma_G \circ \varphi = \gamma_G\}$.

3.2 Weisfeiler-Leman Refinement on Colored Groups

In [4], three versions of Weisfeiler-Leman algorithms on groups were defined. For us it is sufficient to consider two of these versions. The relevant definitions and results are discussed below. They are essentially taken from [4], but we added colorings.

For $k \geq 2$ we devise a *Weisfeiler-Leman algorithm of dimension k (k -WL)* that takes as input a colored group (G, γ) and computes an $\text{Aut}_{\gamma}(G)$ -invariant coloring on $G^{(k)}$. The algorithm computes an initial coloring from isomorphism invariant properties of k -tuples and then iteratively refines color classes until the process stabilizes. The *stable colorings* arising from k -WL provide (possibly incomplete) polynomial-time non-isomorphism tests.

Version I (k -WL_I). The initial coloring $\chi_{\gamma,0}^{I,k}$ is defined via the group's multiplication relation while also taking into account element colors. Two tuples $\bar{g} := (g_1, \dots, g_k)$ and $\bar{h} := (h_1, \dots, h_k)$ obtain the same initial color if and only if for all indices i, j , and m between 1 and k it holds $\gamma(g_i) = \gamma(h_i)$, $g_i = g_j \iff h_i = h_j$, and $g_i g_j = g_m \iff h_i h_j = h_m$. The subsequent refinements are defined iteratively via $\chi_{\gamma,i+1}^{I,k}(\bar{g}) := \left(\chi_{\gamma,i}^{I,k}(\bar{g}), \mathcal{M}(\bar{g}) \right)$. Here, $\mathcal{M}(\bar{g})$ is the multiset of k -tuples of colors given by $\mathcal{M}(\bar{g}) := \{ \{ \chi_{\gamma,i}^{I,k}(\bar{g}_{1 \leftarrow x}), \dots, \chi_{\gamma,i}^{I,k}(\bar{g}_{k \leftarrow x}) \} \mid x \in G \}$, where $\bar{g}_{j \leftarrow x}$ is obtained by replacing the j -th entry of \bar{g} by x .

Version II (k -WL_{II}). The initial coloring $\chi_{\gamma,0}^{II,k}$ is defined in terms of *colored, ordered isomorphism* of tuples. Thus, $\bar{g} = (g_1, \dots, g_k)$ and $\bar{h} = (h_1, \dots, h_k)$ obtain the same initial color if and only if there exists an isomorphism of colored subgroups $\varphi : \langle \bar{g} \rangle \rightarrow \langle \bar{h} \rangle$ such that $\varphi(g_i) = h_i$ for all i . The refinement step is unchanged from Version I.

For finite G there is a smallest i such that $\chi_{\gamma,i}^{I,k}$ and $\chi_{\gamma,i+1}^{I,k}$ induce the same color class partition on $G^{(k)}$. At this point color classes become stable and we obtain the *stable coloring* $\chi_{\gamma}^{I,k} := \chi_{\gamma,i}^{I,k}$. Define $\chi_{\gamma}^{II,k}$ analogously. For uncolored groups write $\chi_G^{I,k}$ and $\chi_G^{II,k}$, respectively.

By definition, the initial colorings are invariant under isomorphisms that respect γ . This property then holds for the iterated colorings as well. In particular, whenever (G, γ_G) and (H, γ_H) are isomorphic as colored groups, there is a bijection $f : G^{(k)} \rightarrow H^{(k)}$ such that $\chi_{\gamma_G}^{I,k} = \chi_{\gamma_H}^{I,k} \circ f$ (and the same holds for Version II). So we obtain a non-isomorphism test by comparing stable colorings computed by k -WL_I or k -WL_{II} as follows.

► **Definition 3.2.** Let (G, γ_G) and (H, γ_H) be colored groups. We say G is distinguished from H by k -WL_I if there is no bijection $f : G^{(k)} \rightarrow H^{(k)}$ with $\chi_{\gamma_G}^{I,k} = \chi_{\gamma_H}^{I,k} \circ f$. We say k -WL_I identifies G if it distinguishes G from all other (non-isomorphic) groups. We write $G \equiv_k^1 H$ to indicate that G and H are not distinguished by k -WL_I. Furthermore, for $m \leq k$, tuples of group elements $\bar{g} \in G^{(m)}$ and $\bar{h} \in H^{(m)}$ are distinguished by k -WL_I if they obtain different colors in the respective induced m -colorings $(\chi_{\gamma_G}^{I,k})^{(m)}$ and $(\chi_{\gamma_H}^{I,k})^{(m)}$. All definitions also apply to Version II in the obvious way.

The different versions of WL on groups are closely related: in particular, $(k+1)$ -WL_I subsumes k -WL_{II}. For the colored versions this is briefly discussed in Lemma 3.4 below.

Finally, we note that in [4], a run time bound of $\mathcal{O}(|G|^{k+1} \log(|G|))$ is given for both versions of k -WL to compute the stable coloring on $G^{(k)}$. The same bound applies to colored groups. In particular, the initial coloring of k -WL_{II} is efficiently computable, since we only have to compute isomorphism types of k -generated subgroups *relative* to a fixed and ordered generating set of size k .

3.3 Bijective k -Pebble Games

As with graphs and uncolored groups, WL-algorithms on colored groups can be characterized via pebble games. For details we refer to the full version (contained in the appendix).

► **Lemma 3.3** (see [4, Theorem 3.2]). Let $J \in \{I, II\}$ and $k \geq 2$. Consider colored groups (G, γ_G) and (H, γ_H) with $\bar{g} \in G^{(k)}$ and $\bar{h} \in H^{(k)}$. Then $\chi_{\gamma_G}^{J,k}(\bar{g}) = \chi_{\gamma_H}^{J,k}(\bar{h})$ if and only if Spoiler has a winning strategy in the configuration $[(g_1, \dots, g_k, \perp), (h_1, \dots, h_k, \perp)]$ in the $(k+1)$ -pebble game (Version J).

In [4] (see [5, Section 3]), relationships for the different versions of WL for uncolored groups are discussed, for the convenience of the reader we sketch the corresponding statement for colored groups here.

► **Lemma 3.4.** *Let (G, γ_G) and (H, γ_H) be colored groups.*

1. *Consider $\bar{g} \in G^{(m)}$, $\bar{h} \in H^{(m)}$ and $k \geq m$. If \bar{g} is distinguished from \bar{h} by k -WL_I then \bar{g} is distinguished from \bar{h} by k -WL_{II}. If \bar{g} is distinguished from \bar{h} by k -WL_{II} then \bar{g} is distinguished from \bar{h} by $(k+1)$ -WL_I.*
2. *It holds that $(G, \gamma_G) \equiv_{k+1}^I (H, \gamma_H) \implies (G, \gamma_G) \equiv_k^{II} (H, \gamma_H) \implies (G, \gamma_G) \equiv_k^I (H, \gamma_H)$.*

Proof Sketch. Part 2) follows from Part 1). For Part 1), the first claim is true by definition. For the second claim, using Lemma 3.4, we compare the $(k+1)$ -pebble game (ver. II) and the $(k+2)$ -pebble game (ver. I) with initial configurations given by placing pebble pairs on (g_i, h_i) for all i . In the version I game, Spoiler copies the winning strategy from the version II game. By assumption, Spoiler eventually reaches a winning configuration in the version II game, meaning that the pebble pairs in this eventual configuration induce a map that does not extend to an isomorphism between the subgroups generated by the respectively pebbled group elements. Then, for each bijection Duplicator may further choose, there must be a witness of this fact, i.e., a word over the currently pebbled group elements in G that is not mapped multiplicatively by Duplicator's bijection. Spoiler can use the extra pebble to win immediately or reduce the length of the witness. A very similar argument is spelled out in the proof of [5, Lemma 3.10, Part 3)] in full detail. ◀

3.4 Induced Colorings & Refinements

We say that a coloring $\gamma_2 : G^{(k)} \rightarrow \mathcal{C}_2$ *refines* a coloring $\gamma_1 : G^{(k)} \rightarrow \mathcal{C}_1$, denoted $\gamma_2 \preceq \gamma_1$, if each γ_1 -color class is a union of γ_2 -color classes.

► **Lemma 3.5.** *Let γ_1, γ_2 be colorings on G such that $(\chi_{\gamma_1}^{I,k})^{(G)} \preceq \gamma_2 \preceq \gamma_1$. Then $\chi_{\gamma_1}^{I,k}$ and $\chi_{\gamma_2}^{I,k}$ induce the same color classes on $G^{(k)}$.*

4 WL-Refinement on Quotient Groups

We investigate the interplay between WL and basic group structure, e.g., subgroups, normal closures or quotients. We use *subset selectors* to compare substructures of different groups.

► **Definition 4.1.** *A subset selector \mathcal{S} associates with each colored group (G, γ) a subset $\mathcal{S}(G, \gamma) \subseteq G$. For each version $J \in \{I, II\}$, a subset selector \mathcal{S} is called k -WL_J-detectable, if $\chi_{\gamma_G}^{J,k}(\mathcal{S}(G, \gamma_G)) \cap \chi_{\gamma_H}^{J,k}(H \setminus \mathcal{S}(H, \gamma_H)) = \emptyset$ holds for all pairs of colored groups $(G, \gamma_G), (H, \gamma_H)$.*

When the dependency of $\mathcal{S}(G, \gamma_G)$ on (G, γ_G) is clear from the context, we also say that $\mathcal{S}(G, \gamma_G)$ is k -WL_J-detectable (instead of $(G, \gamma) \mapsto \mathcal{S}(G, \gamma)$ being detectable). Examples of 2-WL_J-detectable subset selectors include the association of every group with its center ($J = II$) or the subset selector associating with each group the subset of elements of order 2.

We should remark that in our sense detectable means that the subset of interest is a union of $\chi_{\gamma_G}^{J,k}$ -color classes, but we make no statement on how to algorithmically determine which color classes form the set. It might a priori not be clear that the subset is even computable.

If \mathcal{S} is k -WL_J-detectable then $\mathcal{S}(G, \gamma_G)$ is $\chi_{\gamma_G}^{J,k}$ -induced, hence $\text{Aut}_{\gamma_G}(G)$ -invariant. If \mathcal{S} and \mathcal{T} are k -WL_J-detectable, so are their union (intersection) in G and $G \setminus \mathcal{S}(G, \gamma_G)$.

► **Definition 4.2.** *A group expression $\mathcal{E} := (\mathcal{S}_1, \dots, \mathcal{S}_t; \mathcal{R})$ of length t is a sequence of subset selectors \mathcal{S}_i together with a set \mathcal{R} of words $w(x_1, \dots, x_t)$ over t variables x_1, \dots, x_t , allowing inverses. Let (G, γ) be a colored group, then a t -tuple $(g_1, \dots, g_t) \in G^{(t)}$ is a solution to \mathcal{E} if for each i it holds that $g_i \in \mathcal{S}_i(G, \gamma)$ and for each $w \in \mathcal{R}$ it holds that $w(g_1, \dots, g_t) = 1$. Let $\text{Sol}_{\mathcal{E}}(G, \gamma) \subseteq G^{(t)}$ denote the set of all solutions to \mathcal{E} over (G, γ) .*

► **Lemma 4.3.** Consider a group expression $\mathcal{E} := (\mathcal{S}_1, \dots, \mathcal{S}_t; \mathcal{R})$. Let $k \geq t$ and assume that each \mathcal{S}_i is k - WL_{II} -detectable.

1. Let (G, γ_G) and (H, γ_H) be colored groups. Then all t -tuples in $Sol_{\mathcal{E}}(G, \gamma_G)$ can be distinguished from all t -tuples in $H^{(t)} \setminus Sol_{\mathcal{E}}(H, \gamma_H)$ via k - WL_{II} .
2. For $1 \leq j \leq t$ and colored groups (G, γ) define

$$Sol_j^{\exists}(G, \gamma) := \{x \in G \mid \exists(x_1, \dots, x_t) \in Sol_{\mathcal{E}}(G, \gamma) : x_j = x\}$$

$$Sol_j^{\forall}(G, \gamma) := \{x \in G \mid (\forall x_i \in \mathcal{S}_i(G, \gamma))_{1 \leq i \leq t} : (x_1, \dots, x_{j-1}, x, x_{j+1}, \dots, x_t) \in Sol_{\mathcal{E}}(G, \gamma)\}.$$

Then Sol_j^{\exists} and Sol_j^{\forall} are k - WL_{II} -detectable subset selectors for all j .
The same holds for k - WL_I , provided $k > t$.

We can now argue that WL is powerful enough to incorporate various basic group theoretic concepts. This in particular includes generated subgroups, normal closures, powers, conjugacy classes, centralizers, and normalizers. All these statements are relative to inductively detected structures, so the processes can be iterated. Let us record this in the following lemma.

► **Lemma 4.4.** Consider k - WL_{II} -detectable subset selectors \mathcal{S}, \mathcal{T} . Then the following subset selectors are k - WL_{II} -detectable:

1. \mathcal{S}^e for each $e \in \mathbb{Z}$, where $\mathcal{S}^e(G, \gamma) := \{s^e \mid s \in \mathcal{S}(G, \gamma)\}$,
 2. $C_{\mathcal{S}}(\mathcal{T})$, where $C_{\mathcal{S}}(\mathcal{T})(G, \gamma) := \{s \in \mathcal{S}(G, \gamma) \mid [s, \mathcal{T}(G, \gamma)] = \{1\}\}$.
- Provided k is at least 3, k - WL_{II} further detects the following subset selectors:
3. $\{s_1 \dots s_e \mid s_i \in \mathcal{S}(G, \gamma)\}$ for each $e \in \mathbb{N}$, in particular also $\langle \mathcal{S}(G, \gamma) \rangle$,
 4. $\{s^t := tst^{-1} \mid s \in \mathcal{S}(G, \gamma), t \in \mathcal{T}(G, \gamma)\}$, hence the normal closure $\langle \mathcal{S}(G, \gamma)^G \rangle$,
 5. $\mathcal{N}_{\mathcal{S}}(\mathcal{T})$, where $\mathcal{N}_{\mathcal{S}}(\mathcal{T})(G, \gamma) := \{s \in \mathcal{S}(G, \gamma) \mid \mathcal{T}(G, \gamma)^s = \mathcal{T}(G, \gamma)\}$,
 6. $[\mathcal{S}, \mathcal{T}]$, where $[\mathcal{S}, \mathcal{T}](G, \gamma) := \langle [s, t] \mid s \in \mathcal{S}(G, \gamma), t \in \mathcal{T}(G, \gamma) \rangle$.

All statements remain true if we replace Version II by Version I everywhere (including the assumptions), provided $k > 2$ in Parts 1 and 2 and $k > 3$ in Parts 3–6.

We point out how to identify groups as direct products of detectable subgroups.

► **Example 4.5.** Let $G \equiv_3^{\text{II}} H$ and assume that $G = G_1 \times G_2$ with $\chi_G^{\text{II},3}$ -induced subgroups G_i . We use element-colors in G_i to define a detectable subset selector $K \mapsto K_i := \{x \in K \mid \chi_K^{\text{II},k}(x) \in \chi_G^{\text{II},k}(G_i)\}$. Since $G \equiv_3^{\text{II}} H$, also $H_i \equiv_3^{\text{II}} G_i$. By the previous lemma, 3- WL_{II} detects $[G_1, G_2]$ and $G_1 \cap G_2$, which are both trivial, as well as $\langle G_1, G_2 \rangle$, which is equal to G . By definition of detectability, the same must hold for H_1 and H_2 , thus $H = H_1 \times H_2$.

In Section 6 we discuss the (much harder) case of arbitrary direct decompositions, without the assumption that each direct factor is detectable as a subgroup.

Next, we prove that WL is capable of exploiting properties of quotients over detectable subgroups. Later, this can be inductively leveraged along chains of subgroups.

► **Definition 4.6.** Given a coloring $\gamma : G \rightarrow \mathcal{C}$ and a normal subgroup $N \trianglelefteq G$ define the induced quotient coloring $\bar{\gamma}$ on G/N via $\bar{\gamma}(gN) := \{\{\gamma(gn) \mid n \in N\}\}$.

► **Lemma 4.7.** Let $k \geq 4$ and consider colored groups (G, γ_G) and (H, γ_H) . Assume that there are normal subgroups $N_G \trianglelefteq G$ and $N_H \trianglelefteq H$ which are induced by γ_G and γ_H , respectively, such that $\gamma_G(N_G) = \gamma_H(N_H)$. Then

$$\chi_{\bar{\gamma}_G}^{\text{I},k}(g_1 N_G, \dots, g_k N_G) \neq \chi_{\bar{\gamma}_H}^{\text{I},k}(h_1 N_H, \dots, h_k N_H) \implies \chi_{\gamma_G}^{\text{I},k}(g_1, \dots, g_k) \neq \chi_{\gamma_H}^{\text{I},k}(h_1, \dots, h_k)$$

for all choices of $g_i \in G$ and $h_i \in H$.

Proof sketch. The idea is to simulate the pebble game on quotient groups in the pebble game on G and H . Spoiler can first win modulo N_G and N_H , respectively, and then use a constant number of pebbles to manipulate the configuration into one that fulfills the winning condition over the original groups. For details, we refer to the full version. ◀

We synthesize the previous results into our first main theorem, stating that whenever $G \equiv_k^I H$ holds, there is a color preserving correspondence between detectable substructures.

► **Theorem 4.8.** *Let k be at least 4.*

1. *Consider subset selectors N, U and U/N such that for all (G, γ) it holds that $N(G, \gamma) \trianglelefteq G$, $N(G, \gamma) \leq U(G, \gamma)$ and $U/N(G/N(G), \bar{\gamma}) = U(G)/N(G)$. If N and U/N are k - WL_I -detectable then so is U .*
2. *Consider colored groups $(G, \gamma_G) \equiv_k^I (H, \gamma_H)$. Let $\Psi : G \rightarrow H$ be a bijection with $(\chi_{\gamma_G}^{I,k})^{(G)} \circ \Psi = (\chi_{\gamma_H}^{I,k})^{(H)}$. Then $M \subseteq G$ is $\chi_{\gamma_G}^{I,k}$ -induced if and only if $\Psi(M) \subseteq H$ is $\chi_{\gamma_H}^{I,k}$ -induced. In this case it holds that $\Psi(\langle M \rangle) = \langle \Psi(M) \rangle$. In particular, if M is a subgroup then so is $\Psi(M)$ and it holds $(M, \gamma_G|_M) \equiv_k^I (\Psi(M), \gamma_H|_{\Psi(M)})$. Additionally, M is normal if and only if $\Psi(M)$ is and then it also holds that $(G/M, \bar{\gamma}_G) \equiv_k^I (H/\Psi(M), \bar{\gamma}_H)$.*

Finally let us point out that detectable substructures can be used to limit Duplicator-strategies. This technique will be needed towards the main result of Section 6. More precisely, we show that Spoiler can “trade off” one pebble pair to enforce that Duplicator’s bijections are simultaneously compatible with detectable substructures in the following sense.

► **Lemma 4.9.** *Let $k \geq 3$ and $J \in \{I, II\}$. Consider groups G and H with $G \equiv_k^J H$, so Duplicator has a winning strategy in the $(k+1)$ -pebble game (Version J). Assume $\chi_G^{J,k}$ and $\chi_H^{J,k}$ induce chains of subgroups $G_s \leq \dots \leq G_1 \leq G$ and $H_s \leq \dots \leq H_1 \leq H$, respectively, such that $\chi_G^{J,k}(G_i) = \chi_H^{J,k}(H_i)$ for all i . Then Duplicator has a winning strategy in the k -pebble game (Version J) on (G, H) such that each bijection $f : G \rightarrow H$ chosen by Duplicator’s strategy fulfills the following condition: $\forall x \in G \forall i : f(xG_i) = f(x)H_i$.*

The proof actually works in a context more general than groups, replacing subgroup chains by nested equipartitions. This generalization might find applications in different contexts.

5 WL-dimension of certain isomorphism invariants

We just briefly summarize our results in what follows. A detailed treatment can be found in the full version, for group theoretic foundations see for example [16].

► **Lemma 5.1.** *For $k \geq 2$, k - WL_{II} identifies all finite k -generated groups and all finite abelian groups.*

5.1 Derived & Central Series

► **Lemma 5.2.**

1. *For $k \geq 3$, $G' := [G, G]$ is k - WL_{II} -detectable.*
2. *Assume that $k \geq 4$ and $G \equiv_k^I H$ hold. Let $G_0 \geq G_1 \geq \dots \geq G_t$ denote the derived, upper central or lower central series of G (without redundancies, starting at $G_0 := G$). Define the corresponding series of H via $H_0 \geq \dots \geq H_s$. Then $s = t$ holds and for all i we have that $G_i \equiv_k^I H_i$, as well as $G_i/G_{i+1} \cong H_i/H_{i+1}$.*

27:10 Isomorphism Invariants of Groups and the WL-Dimension

The requirement $k \geq 3$ is necessary in the first statement as computations on `SmallGroup(128,171)` and `SmallGroup(128,1122)` from the Small Groups Library in GAP [14] show.

► **Corollary 5.3.** *For $k \geq 4$, k -WL_I distinguishes solvable from non-solvable groups and k -WL_I distinguishes between groups of different nilpotency classes.*

5.2 Radicals

Let \mathcal{F} be a class of finite groups that is closed under isomorphism and normal products. Then the \mathcal{F} -radical $\mathcal{O}_{\mathcal{F}}(G)$ of G is defined as the largest normal \mathcal{F} -subgroup in G .

► **Lemma 5.4.** *Let $k \geq 3$. If \mathcal{F} is closed under normal subgroups and $(k-1)$ -WL_{II} distinguishes \mathcal{F} -groups from all non- \mathcal{F} -groups, then k -WL_{II} detects $\mathcal{O}_{\mathcal{F}}(G)$ in G .*

► **Lemma 5.5.** *The solvable radical is 2-WL_{II}-detectable. The nilpotent radical $\text{Fit}(G)$ and all π -radicals $\mathcal{O}_{\pi}(G)$ (π a collection of primes) are 3-WL_{II}-detectable.*

► **Lemma 5.6.** *If G contains a unique maximal abelian normal subgroup, then it is 3-WL_{II}-detectable.*

5.3 Simple Groups & Composition Factors

Recall that finite (almost) simple groups can be generated with 2 (respectively 3) elements [9].

► **Lemma 5.7.** *2-WL_{II} identifies finite simple groups. 3-WL_{II} identifies finite almost simple groups and finite direct products of simple groups.*

In the case of simple groups there is a stronger result, stating that simple groups are uniquely identified among all groups up to isomorphism by their order and the orders of their elements [27].

► **Theorem 5.8.** *The socle of a finite group G is 4-WL_{II}-detectable. Let $k \geq 5$ and $G \cong_k H$, then G and H have the same composition factors (with multiplicities).*

6 WL-Refinement and Direct Products

In this final section we study the detectability of direct product structures in finite groups. The section is organized similar to [19], in the sense that we first consider direct products where one factor is an abelian group (the semi-abelian case) and reduce to these in the general case later on. A crucial difference between our setting and the one in [19] is that in the latter, computations can be executed as long as they are efficient, where in our case, we are analyzing a fixed algorithm that cannot make non-canonical choices.

► **Definition 6.1.** *A group G is the (internal) central product of subgroups $G_1, G_2 \leq G$, if it holds that $G = \langle G_1, G_2 \rangle$ and $[G_1, G_2] = \{1\}$.*

Our main difficulty is that a group can admit several inherently different central decompositions. In contrast to that recall that indecomposable *direct* decompositions are unique in the following sense.

► **Lemma 6.2.** *Let $G = G_1 \times \cdots \times G_m = H_1 \times \cdots \times H_n$ be two decompositions of G into directly indecomposable factors. Then $n = m$ and there is a permutation $\sigma \in S_m$ such that for all i we have $G_i \cong H_{\sigma(i)}$ and $G_i Z(G) = H_{\sigma(i)} Z(G)$.*

Proof. The first part is the well-known Krull-Remak-Schmidt Theorem and the addition that $G_i Z(G) = H_{\sigma(i)} Z(G)$ can be easily derived (see for example [19, Corollary 6]) ◀

In particular, the collection of subgroups $\{G_i Z(G)\}_{1 \leq i \leq m}$ is invariant under automorphisms as a whole. Later we show that $\bigcup_{i=1}^m G_i Z(G)$ is 5-WL₁-detectable.

► **Lemma 6.3.** *If $J \in \{I, II\}$, $k \geq 3$, $G_1 \cong_k^J H_1$ and $G_2 \cong_k^J H_2$, then $G_1 \times G_2 \cong_k^J H_1 \times H_2$.*

The opposite direction is investigated below and turns out to be highly non-trivial.

6.1 Abelian and Semi-Abelian Case

Direct products with abelian groups serve as a basis for reduction later on.

► **Definition 6.4.** *An element $x \in G$ splits from G if there is a complement $H \leq G$ of x in G , i.e., $G = \langle x \rangle \times H$.*

A detailed treatment of splitting elements can be found in the full version.

► **Corollary 6.5.** *The set of elements splitting from a finite group is 4-WL₁-detectable.*

The splitting of elements can reveal information about direct decompositions of a group.

► **Lemma 6.6.** *Consider a direct product $G = G_1 \times G_2$ and a p -element $z := (z_1, z_2) \in Z(G)$. Then z splits from G if and only if z_i splits from G_i for some $i \in \{1, 2\}$ which fulfills $|z_i| = |z|$.*

This can be inductively leveraged to handle the semi-abelian case, by which we mean groups of the form $H \times A$ where A is abelian and H does not have abelian direct factors.

► **Lemma 6.7.** *Let $G = H \times A$ with A a maximal abelian direct factor. The isomorphism type of A is identified by 4-WL₁, i.e., if $\hat{G} \cong_4^1 G$ then \hat{G} has a maximal abelian direct factor isomorphic to A .*

Controlling the non-abelian part is more complicated and led us to introduce a new technical framework.

► **Definition 6.8.** *Let $G = L \times R$. A component-wise filtration of $U \leq G$ w.r.t. L and R is a chain of subgroups $\{1\} = U_0 \leq \dots \leq U_r = U$ such that for all $1 \leq i < r$, we have $U_{i+1} \leq U_i(L \times \{1\})$ or $U_{i+1} \leq U_i(\{1\} \times R)$. The filtration is k -WL₁-detectable if all subgroups in the chain are.*

► **Lemma 6.9.** *Let $G = H \times A$ with maximal abelian direct factor A . There exists a component-wise filtration of $Z(G)$ with respect to H and A that is 4-WL₁-detectable.*

► **Lemma 6.10.** *Consider $G := H \times A$ and $\hat{G} = \hat{H} \times \hat{A}$ where A and \hat{A} are maximal abelian direct factors. Then, for $k \geq 5$, $G \cong_k^1 \hat{G}$ implies $H \cong_{k-1}^1 \hat{H}$.*

6.2 General Case

The general case is reduced to the semi-abelian case. Consider an indecomposable direct decomposition $G = G_1 \times \dots \times G_d$. We first show that $\bigcup_i G_i Z(G)$ can be detected by WL and then we exploit the fact that the non-commuting graph induces components on $\bigcup_i G_i Z(G)$ which correspond to the groups $G_i Z(G)$.

► **Definition 6.11.** *Given a group G , we define the non-commuting graph Γ_G with vertex set G , in which two elements $g, h \in G$ are joined by an edge if and only if $[g, h] \neq 1$.*

► **Lemma 6.12** ([1], Prop. 2.1). *If G is non-abelian then $\Gamma_G[G \setminus Z(G)]$ is connected.*

We now approximate $\bigcup_i G_i Z(G)$ from below by constructing a canonical central decomposition of G which is WL-detectable.

► **Definition 6.13.** *Consider a finite, non-abelian group G . Define $M_1 \subseteq G$ to be the set of non-central elements g whose centralizers $C_G(g)$ have maximal order among all non-central elements. Iteratively define M_{i+1} by adding those elements g to M_i that have maximal centralizer order $|C_G(g)|$ among the remaining elements $G \setminus \langle M_i \rangle$. Set $M := M_\infty$ to be the stable set resulting from this process. Consider the subgraph of Γ_G induced on M and let K_1, \dots, K_m be its connected components. Set $N_i := \langle K_i \rangle$. We call N_1, \dots, N_m the non-abelian components of G .*

► **Lemma 6.14.** *In the notation of the previous definition, the following hold:*

1. M is detectable in G by 3-WL_{II}.
2. $G = N_1 \cdots N_m$ is a central decomposition of G . For all i , $Z(G) \leq N_i$ and N_i is non-abelian. In particular M generates G .
3. If $G = G_1 \times \cdots \times G_d$ is an arbitrary direct decomposition, then for each $1 \leq i \leq m$ there is exactly one $1 \leq j \leq d$ with $N_i \subseteq G_j Z(G)$. Collect all such i for one fixed j in an index set I_j . Then the product over all N_i for $i \in I_j$ is equal to $G_j Z(G)$.

► **Definition 6.15.** *Let $G = N_1 \cdots N_m$ be the decomposition into non-abelian components and let $G = G_1 \times \cdots \times G_d$ be an arbitrary direct decomposition. We say $x \in G$ is full for $(G_{j_1}, \dots, G_{j_r})$, if $\{1 \leq i \leq m \mid [x, N_i] \neq 1\} = I_{j_1} \cup \cdots \cup I_{j_r}$. For all $x \in G$ define $C_x := \prod_{[x, N_i] = \{1\}} N_i$ and $N_x := \prod_{[x, N_i] \neq \{1\}} N_i$.*

Overall, when grouped adequately, the full elements with maximal centralizers generate the direct factors modulo central elements (see full version).

► **Lemma 6.16.** *Let $G = N_1 \cdots N_m$ be the decomposition into non-abelian components and $G = G_1 \times \cdots \times G_d$ a decomposition into indecomposable direct factors. For $k \geq 5$, k -WL_{II} detects the set of elements that are full for only one G_i as well as the pairs of elements that are full for the same collection of direct factors.*

► **Corollary 6.17.** *If $G = G_1 \times \cdots \times G_d$ is a decomposition into indecomposable direct factors then $\bigcup_i G_i Z(G)$ is detected in G by 5-WL_{II}.*

► **Theorem 6.18.** *Let $G = G_1 \times \cdots \times G_d$ be a decomposition into indecomposable direct factors and $k \geq 5$. If $G \cong_k^{\Pi} H$ then there are indecomposable direct factors $H_i \leq H$ such that $H = H_1 \times \cdots \times H_d$ and $G_i \cong_{k-1}^{\Pi} H_i$ for all i . Moreover G and H have isomorphic maximal abelian direct factors and $G_i Z(G) \cong_k^{\Pi} H_i Z(H)$.*

Proof. Since $\mathcal{F}_G := \bigcup_i G_i Z(G)$ is 5-WL_{II}-detectable, the group H must be decomposable into indecomposable direct factors $H = \times_j H_j$ such that $\mathcal{F}_H = \bigcup_j H_j Z(H) \subseteq H$ is indistinguishable from \mathcal{F}_G . Consider the non-commuting graphs of G and H induced on these sets and recall that non-commuting graphs of non-abelian groups are connected (Lemma 6.12). Since different direct factors in a fixed decomposition centralize each other, we obtain that for each non-singleton connected component K of $\Gamma_G[\mathcal{F}_G]$ there exists a unique indecomposable direct factor G_i such that $K = G_i Z(G) \setminus Z(G)$ and thus $\langle K \rangle = G_i Z(G)$. Again by Lemma 6.12, all non-abelian direct factors appear in this way.

The same holds for H and so if G is not distinguishable from H , there must be a bijection between the components of $\Gamma_G[\mathcal{F}_G]$ and $\Gamma_H[\mathcal{F}_H]$, such that the subgroups generated by corresponding components are indistinguishable via 5-WL_{II}. This defines a correspondence $G_i Z(G) \cong_k^{\Pi} H_i Z(H)$ after reordering the factors of H in an appropriate way. From Lemma 6.10 it follows that $G_i \cong_{k-1}^{\Pi} H_i$. By Lemma 6.9, G and H must have isomorphic maximal abelian direct factors, so for abelian factors we even have $G_i \cong H_i$. ◀

7 Conclusion

We studied the Weisfeiler-Leman dimension of numerous isomorphism invariants of groups, showing that a low dimensional WL-algorithm in fact captures a plethora of isomorphism invariants, characteristic subgroups, and group properties classic to algorithmic group theory. Particularly tricky was the treatment of direct indecomposable factors, for which we had to circumvent the fact that they do not correspond to canonical substructures of the groups. Our techniques lead us to a canonical maximal central decomposition.

The observation that many efficiently computable isomorphism invariants are captured by a low dimensional WL-algorithm raises the question whether there are actually invariants that are not captured at all. Here we should emphasize that it is an open problem whether some fixed dimension of WL represents a complete invariant. The question is equivalent to the well-known open question whether the Weisfeiler-Leman dimension of groups is bounded in general (stated explicitly in [4]).

For this open question, our results show that it suffices to consider directly indecomposable groups. We wonder whether there are other, similar reductions to confine the search for groups of high WL-dimension.

References

- 1 Alireza Abdollahi, Saieed Akbari, and Hamid R. Maimani. Non-commuting graph of a group. *J. Algebra*, 298(2):468–492, April 2006. doi:10.1016/j.jalgebra.2006.02.015.
- 2 Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On Weisfeiler-Leman invariance: Subgraph counts and related graph properties. *J. Comput. Syst. Sci.*, 113:42–59, 2020. doi:10.1016/j.jcss.2020.04.003.
- 3 Hans Ulrich Besche and Bettina Eick. Construction of finite groups. *J. Symb. Comput.*, 27(4):387–404, 1999. doi:10.1006/jscs.1998.0258.
- 4 Jendrik Brachter and Pascal Schweitzer. On the Weisfeiler-Leman dimension of finite groups. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '20*, pages 287–300, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394786.
- 5 Jendrik Brachter and Pascal Schweitzer. On the Weisfeiler-Leman dimension of finite groups. *CoRR*, abs/2003.13745, 2020. arXiv. arXiv:2003.13745.
- 6 Peter A. Brooksbank, Joshua A. Grochow, Yinan Li, Youming Qiao, and James B. Wilson. Incorporating Weisfeiler-Leman into algorithms for group isomorphism. *CoRR*, abs/1905.02518, 2019. arXiv:1905.02518.
- 7 Arthur Cayley. On the theory of groups, as depending on the symbolic equation $\theta^n = 1$. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 7(42):40–47, 1854. doi:10.1080/14786445408647421.
- 8 Mun See Chang and Christopher Jefferson. Disjoint direct product decompositions of permutation groups. *J. Symb. Comput.*, 108:1–16, 2022. doi:10.1016/j.jsc.2021.04.003.
- 9 Francesca Dalla Volta and Andrea Lucchini. Generation of almost simple groups. *J. Algebra*, 178(1):194–223, 1995. doi:10.1006/jabr.1995.1345.
- 10 Bireswar Das and Shivdutt Sharma. Nearly linear time isomorphism algorithms for some nonabelian group classes. In René van Bevern and Gregory Kucherov, editors, *Computer Science - Theory and Applications - 14th International Computer Science Symposium in Russia, CSR 2019, Novosibirsk, Russia, July 1-5, 2019, Proceedings*, volume 11532 of *Lecture Notes in Computer Science*, pages 80–92. Springer, 2019. doi:10.1007/978-3-030-19955-5_8.
- 11 Heiko Dietrich and James B. Wilson. Polynomial-time isomorphism testing of groups of most finite orders. *CoRR*, abs/1806.08872, 2018. arXiv. arXiv:1806.08872.

- 12 Bettina Eick and Max Horn. The construction of finite solvable groups revisited. *J. Algebra*, 408:166–182, 2014. doi:10.1016/j.jalgebra.2013.09.028.
- 13 Bettina Eick, Max Horn, and Alexander Hulpke. Constructing groups of ‘small’ order: Recent results and open problems. In Gebhard Böckle, Wolfram Decker, and Gunter Malle, editors, *Algorithmic and Experimental Methods in Algebra, Geometry, and Number Theory*, pages 199–211. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-70566-8_8.
- 14 The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.11.1*, 2021. URL: <https://www.gap-system.org>.
- 15 Joshua A. Grochow and Youming Qiao. On the complexity of isomorphism problems for tensors, groups, and polynomials I: tensor isomorphism-completeness. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPICs*, pages 31:1–31:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ITCS.2021.31.
- 16 Marshall Hall. *The Theory of Groups*. AMS Chelsea Publishing Series. AMS Chelsea Pub., 1999. URL: <https://books.google.de/books?id=oyxnWF9ssI8C>.
- 17 Neil Immerman. Relational queries computable in polynomial time. *Inf. Control.*, 68(1-3):86–104, 1986. doi:10.1016/S0019-9958(86)80029-8.
- 18 Telikepalli Kavitha. Linear time algorithms for abelian group isomorphism and related problems. *J. Comput. Syst. Sci.*, 73(6):986–996, 2007. doi:10.1016/j.jcss.2007.03.013.
- 19 Neeraj Kayal and Timur Nezhmetdinov. Factoring groups efficiently. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikolettseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 585–596. Springer, 2009. doi:10.1007/978-3-642-02927-1_49.
- 20 Sandra Kiefer. *Power and limits of the Weisfeiler-Leman algorithm*. PhD thesis, RWTH Aachen University, 2020.
- 21 Eugene M. Luks. Group isomorphism with fixed subnormal chains. *CoRR*, abs/1511.00151, 2015. arXiv:1511.00151.
- 22 Eamonn A. O’Brien. The p-group generation algorithm. *J. Symb. Comput.*, 9(5/6):677–698, 1990. doi:10.1016/S0747-7171(08)80082-X.
- 23 David J. Rosenbaum. Bidirectional collision detection and faster deterministic isomorphism testing. *CoRR*, abs/1304.3935, 2013. arXiv. arXiv:1304.3935.
- 24 David J. Rosenbaum. Breaking the $n^{\log n}$ barrier for solvable-group isomorphism. 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 1054–1073. SIAM, 2013. doi:10.1137/1.9781611973105.76.
- 25 Michael J. Smith. *Computing automorphisms of finite soluble groups*. PhD thesis, Australian National University, 1995.
- 26 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM, 1982. doi:10.1145/800070.802186.
- 27 A. V. Vasil’ev, M. A. Grechkoseeva, and V. D. Mazurov. Characterization of the finite simple groups by spectrum and order. *Algebra and Logic*, 48:385–409, 2009. doi:10.1007/s10469-009-9074-9.
- 28 James B. Wilson. Finding direct product decompositions in polynomial time. *CoRR*, abs/1005.0548, 2010. arXiv. arXiv:1005.0548.

Faster Approximate Covering of Subcurves Under the Fréchet Distance

Frederik Brüning

Department of Computer Science, Universität Bonn, Germany

Jacobus Conradi

Department of Computer Science, Universität Bonn, Germany

Anne Driemel

Hausdorff Center for Mathematics, Universität Bonn, Germany

Abstract

Subtrajectory clustering is an important variant of the trajectory clustering problem, where the start and endpoints of trajectory patterns within the collected trajectory data are not known in advance. We study this problem in the form of a set cover problem for a given polygonal curve: find the smallest number k of representative curves such that any point on the input curve is contained in a subcurve that has Fréchet distance at most a given Δ to a representative curve. We focus on the case where the representative curves are line segments and approach this NP-hard problem with classical techniques from the area of geometric set cover: we use a variant of the multiplicative weights update method which was first suggested by Brönniman and Goodrich for set cover instances with small VC-dimension. We obtain a bicriteria-approximation algorithm that computes a set of $O(k \log(k))$ line segments that cover a given polygonal curve of n vertices under Fréchet distance at most $O(\Delta)$. We show that the algorithm runs in $\tilde{O}(k^2 n + kn^3)$ time in expectation and uses $\tilde{O}(kn + n^3)$ space. For input curves that are c -packed and lie in the plane, we bound the expected running time by $\tilde{O}(k^2 c^2 n)$ and the space by $\tilde{O}(kn + c^2 n)$. In addition, we present a variant of the algorithm that uses implicit weight updates on the candidate set and thereby achieves near-linear running time in n without any assumptions on the input curve, while keeping the same approximation bounds. This comes at the expense of a small (polylogarithmic) dependency on the relative arclength.

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

Keywords and phrases Clustering, Set cover, Fréchet distance, Approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.28

Related Version *Full Version:* <https://arxiv.org/abs/2204.09949> [7]

Funding This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – AA 1111/2-2 (FOR 2535 Anticipating Human Behavior).

1 Introduction

The advancement of tracking technology made it possible to record the movement of single entities at a large scale in various application areas ranging from vehicle navigation over sports analytics to the socio-ecological study of animal and human behaviour. The types of trajectories that are analyzed range from GPS-trajectories [25] to full-body-motion trajectories [22] and complex gestures [24], and even include the positions of the focus point of attention from a human eye [15, 21].

In many such applications, a flood of data presents us with the challenging task of extracting useful information. If a long trajectory is given as a sequence of positions in some parameter space, it is rarely known in advance which specific movement patterns occur. In particular, it is challenging to find the start and endpoints of such patterns, which is why popular clustering algorithms heuristically partition the trajectories into smaller subtrajectories. An example is the popular algorithm by Lee, Han and Whang [23].



© Frederik Brüning, Jacobus Conradi, and Anne Driemel;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 28; pp. 28:1–28:16

Leibniz International Proceedings in Informatics



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

Since the criteria according to which one should detect, group and represent behaviour patterns vary greatly among different kinds of application, there are many different variants of the subtrajectory clustering problem, see also the survey papers [12, 26, 27]. One line of research uses the well-established Fréchet distance to define similarity between subcurves, for example the works of Agarwal et al. [1], Buchin et al. [11] and Akitaya et al. [2].

In an attempt to unify previous definitions of the underlying algorithmic problem, Akitaya et al. [2] define the following geometric set cover problem. Given a polygonal curve, the goal is to “cover” the whole curve with a minimum number of simpler representative curves, such that each point of the trajectory is contained in a subcurve with small Fréchet distance to its closest representative curve. This is in line with traditional clustering formulations such as metric k -center, where clusters may overlap. In this paper, we study the set cover problem introduced by Akitaya et al. and improve upon their results.

Preliminaries. For any $n > 1$, a sequence of points $p_1, \dots, p_n \in \mathbb{R}^d$ defines a **polygonal curve** P by linearly interpolating consecutive points, that is, for each i , we obtain the **edge** $e_i : [0, 1] \rightarrow \mathbb{R}^d; t \mapsto (1 - t)p_i + tp_{i+1}$. We may write $e_i = \overline{p_i p_{i+1}}$ for edges. We may think of P as a continuous function $P : [0, 1] \rightarrow \mathbb{R}^d$ by fixing n values $0 = t_1 < \dots < t_n = 1$, and defining $P(t) = e_i \left(\frac{t - t_i}{t_{i+1} - t_i} \right)$ for $t_i \leq t \leq t_{i+1}$. We call the set (t_1, \dots, t_n) the **vertex parameters** of the parametrized curve $P : [0, 1] \rightarrow \mathbb{R}^d$. For $n = 1$, we may slightly abuse notation to view a point p_1 in \mathbb{R}^d as a polygonal curve defined by an edge of length zero with $p_2 = p_1$. We call the number of vertices n the **complexity** of the curve. For any two $a, b \in [0, 1]$ we denote with $P[a, b]$ the **subcurve** of P that starts at $P(a)$ and ends at $P(b)$. Note, that $a > b$ is specifically allowed and results in a subcurve in reverse direction. We call the subcurves of edges **subedges**. Let $\mathbb{X}_\ell^d = (\mathbb{R}^d)^\ell$, and think of the elements of this set as the set of all polygonal curves of ℓ vertices in \mathbb{R}^d .

For two parametrized curves P and Q , we define their **Fréchet distance** as

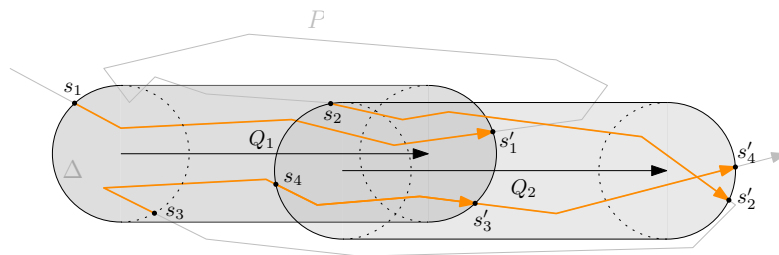
$$d_F(P, Q) = \inf_{\alpha, \beta: [0,1] \rightarrow [0,1]} \sup_{t \in [0,1]} \|P(\alpha(t)) - Q(\beta(t))\|,$$

where α and β range over all functions that are non-decreasing, surjective and continuous. We call the pair (α, β) a traversal. Every traversal has a distance $\sup_{t \in [0,1]} \|P(\alpha(t)) - Q(\beta(t))\|$ associated to it.

We call a curve X in \mathbb{R}^d **c -packed**, if for any point p and and radius r , the length of X inside the disk is bounded by $\|X \cap b_r(p)\| \leq cr$, where $b_r(p) = \{x \in \mathbb{R}^d \mid \|p - x\| \leq r\}$. Let X be a set. We call a set \mathcal{R} where any $r \in \mathcal{R}$ is of the form $r \subseteq X$ a **set system** with **ground set** X .

We say a subset $A \subseteq X$ is **shattered** by \mathcal{R} if for any $A' \subseteq A$ there exists an $r \in \mathcal{R}$ such that $A' = r \cap A$. The **VC-dimension** of \mathcal{R} is the maximal size of a set A that is shattered by \mathcal{R} . For a weight function w on the ground set X and a real value $\varepsilon > 0$, we say that a subset $C \subseteq X$ is an **ε -net** if every set of \mathcal{R} of weight at least $\varepsilon \cdot w(X)$ contains at least one element of C . For any $A \subseteq X$, we write $w(A)$ short for $\sum_{a \in A} w(a)$.

Computational Model. We describe our algorithms in the real-RAM model of computation, which allows to store real numbers and to perform simple operations in constant time on them. We call the following operations **simple operations**. The arithmetic operations $+$, $-$, \times , $/$. The comparison operations $=$, \neq , $>$, \geq , \leq , $<$, for real numbers with output 0 or 1. In addition to the simple operations, we allow the square-root operation. In the full version [7], we describe how to circumvent the square-root operation with little extra cost.



■ **Figure 1** Illustration of the Δ -coverage of a set $C = \{Q_1, Q_2\}$ and a curve P . Here we have $\Psi_\Delta(P, C) = [s_1, s'_1] \cup [s_2, s'_2] \cup [s_3, s'_3] \cup [s_4, s'_4]$, since the subcurves $P[s_1, s'_1]$ and $P[s_3, s'_3]$ have Fréchet distance Δ to Q_1 , the subcurves $P[s_2, s'_2]$ and $P[s_4, s'_4]$ have Fréchet distance Δ to Q_2 and each other subcurve of P that has Fréchet distance at most Δ to Q_1 or Q_2 is a subcurve of $P[s_i, s'_i]$ for some $1 \leq i \leq 4$.

Problem definition. We study the same problem as Akitaya, Chambers, Brünig and Driemel [2]. Let $P : [0, 1] \rightarrow \mathbb{R}^d$ be a polygonal curve of n vertices and let $\ell \in \mathbb{N}$ and $\Delta \in \mathbb{R}$ be fixed parameters. Define the Δ -coverage of a set of center curves $C \subseteq \mathbb{X}_\ell^d$ as follows:

$$\Psi_\Delta(P, C) = \bigcup_{q \in C} \bigcup_{0 \leq t \leq t' \leq 1} \{s \in [t, t'] \mid d_F(P[t, t'], q) \leq \Delta\}.$$

The Δ -coverage corresponds to the part of the curve P that is covered by the set of all subtrajectories that are within Fréchet distance Δ to some curve in C . If for some P, C, Δ it holds that $\Psi_\Delta(P, C) = [0, 1]$, then we call C a Δ -covering of P . The problem is to find a Δ -covering $C \subseteq \mathbb{X}_\ell^d$ of P of minimum size. We study bicriterial approximation algorithms for this problem, which we formalize as follows.

► **Definition 1** ((α, β) -approximate solution). Let $P \in \mathbb{X}_n^d$ be a polygonal curve, $\Delta \in \mathbb{R}_+$ and $\ell \in \mathbb{N}$. A set $C \subseteq \mathbb{X}_\ell^d$ is an (α, β) -approximate solution to the Δ -coverage problem on P , if C is an $\alpha\Delta$ -covering of P and there exists no Δ -covering $C' \subseteq \mathbb{X}_\ell^d$ of P with $\beta|C'| < |C|$.

Related work. Buchin, Buchin, Gudmundsson, Löffler and Luo were the first to consider the problem of clustering subtrajectories under the Fréchet distance [10]. They consider the problem of finding a single cluster of subtrajectories with certain qualities, like the number of distinct subtrajectories, or the length of the longest subtrajectory assigned to it. In their paper, they suggested a swepline approach in the parameter space of the curves and obtain constant-factor approximation algorithms for finding the largest cluster. They also show NP-completeness of the corresponding decision problems. This hardness result extends to $(2 - \varepsilon)$ -approximate algorithms. For their 2-approximation algorithm, Buchin et al. [10] develop an algorithm that finds a legible cluster center among the subcurves of the input curve. Gudmundsson and Wong [19] present a cubic conditional lower bound for this problem and show that it is tight up to a factor of $O(n^{o(1)})$, where n is the number of vertices.

The algorithmic ideas presented in [10] were implemented and extended by Gudmundsson and Valladares [18] who obtained practical speed ups using GPUs. In a series of papers, these ideas were also applied to the problem of reconstructing road maps from GPS data [8, 9]. In a similar vein, Buchin, Kilgus and Kölzsch [11] studied the trajectories of migrating animals and defined so-called group diagrams which are meant to represent the underlying migration patterns in the form of a graph. In their algorithm, to build the group diagram, they repeatedly find the largest cluster and remove it from the data, inspired by the classical greedy set cover algorithm.

The above cited works however do not offer theoretical guarantees when used for computing a clustering of subtrajectories, nor do they explicitly formulate a clustering objective. Agarwal, Fox, Munagala, Nath, Pan, and Taylor [1] define an objective function for clustering subtrajectories based on the metric facility location problem, which consists of a weighted sum over different quality measures such as the number of centers and the distances between cluster centers and their assigned trajectories. While they show NP-hardness for determining whether an input curve can be covered with respect to the Fréchet distance, they also present a $O(\log^2 n)$ -approximation algorithm for clustering κ -packed curves (for some constant κ) under the discrete Fréchet distance, where n denotes the total complexity of the input. The overall running time of their algorithm is roughly quadratic in n , cubic in κ and depends logarithmically on the spread of the vertex coordinates.

In our paper, we focus on the clustering formulation previously studied by Akitaya, Chambers, Brüning, and Driemel [2]. They present a pseudo-polynomial algorithm that computes a bi-criterial approximation in the sense of Definition 1 with expected running time in $\tilde{O}(k(\frac{\lambda}{\Delta})^2 + \frac{\lambda}{\Delta}n)$, where λ denotes the total arclength of the input trajectory. The algorithm finds an (α, β) -approximate solution with $\alpha \in O(1)$ and $\beta \in O(\ell^2 \log(k\ell))$. In combination with our Lemma 2, below, this can be directly improved to $O(\ell \log(k))$. It should be noted that in this problem formulation some complexity constraint on the eligible cluster centers is needed to prevent the entire input curve being a cluster center in a trivial clustering.

Our contribution. Our main result is an algorithm that computes an (α, β) -approximate solution with $\alpha \in O(1)$ and $\beta = O(\ell \log k)$, where k is the size of an optimal solution. For general curves, the algorithm runs in $\tilde{O}(k^2n + kn^3)$ time in expectation and uses $\tilde{O}(kn + n^3)$ space. (The $\tilde{O}(\cdot)$ notation hides polylogarithmic factors in n to simplify the exposition.) If the input curve is a c -packed polygonal curve in the plane, the expected running time can be bounded by $\tilde{O}(k^2c^2n)$ and the space is in $\tilde{O}(kn + c^2n)$. In higher dimensions, the bound for c -packed curves becomes quadratic in n . Our second result is an algorithm that achieves near-linear running time in n – even for general polygonal curves – while keeping the same approximation bounds at the expense of a small dependency on the arclength in the running time. The algorithm needs in expectation $\tilde{O}(nk^3 \log^4(\frac{\lambda}{\Delta k}))$ time and $\tilde{O}(nk \log(\frac{\lambda}{\Delta k}))$ space, where λ is the total arclength of the input curve. Here, we stated our results for general ℓ using the reduction described below (Lemma 2).

In our algorithms we use a variant of the multiplicative weights update method [5], which has been used earlier for set cover problems with small VC-dimension [6, 13]. The difficulty in our case is that the set system initially has high VC-dimension, as shown by Akitaya et al [2] – namely $\Theta(\log n)$ in the worst case. We circumvent this by defining an intermediate set cover problem where the VC-dimension is significantly reduced. We then show how to compute a finite set system using a carefully chosen set of candidate curves on which the multiplicative weight update method can be applied. A key idea that enables our results is a curve simplification that requires the curve to be locally maximally simplified, a notion that is borrowed from de Berg, Cook, and Gudmundsson [14]. To the best of our knowledge, our candidate generation yields the first strongly polynomial algorithm for approximate subtrajectory clustering under the continuous Fréchet distance. In the full version [7], we also discuss how our candidate set can be used for the related problem of maximizing the coverage. Our second algorithm improves the dependency on the relative arclength from quadratic to polylogarithmic as compared to [2].

Reduction to line segments. In the remainder of the paper, we will focus on finding a Δ -covering with line segments, that is $\ell = 2$. The following lemma provides the reduction for general ℓ at the expense of an increased approximation factor.

► **Lemma 2.** *Let $P \in \mathbb{X}_n^d$ be a polygonal curve, $\Delta \in \mathbb{R}_+$ and $\ell \in \mathbb{N}$. Let $C \subseteq \mathbb{X}_\ell^d$ be a Δ -covering of P of minimum cardinality. There exists a set of line segments $C' \subseteq \mathbb{X}_2^d$ that is a Δ -covering of P with $|C'| \leq (\ell - 1)|C|$.*

Proof. Choose as set C' the union of the set of edges of the polygonal curves of C . Clearly, this set has the claimed cardinality and is a Δ -covering of P . ◀

Roadmap. In Section 2 we develop a structured variant of our problem that allows us to apply the multiplicative weight update method in the style of Brönniman and Goodrich [6] in an efficient way. Our intermediate goal is to obtain a structured set of candidates for a modified coverage problem that is on the one hand easy to compute and on the other hand sufficient to obtain good approximation bounds for the original problem. We first define our notion of curve simplification. A crucial property of this simplification is that the relevant subcurves of the input are within small Fréchet distance to subcurves of constant complexity of the simplification. We then define a structured notion of Δ -coverage and a candidate space, which lets us take advantage of this fact. We show that we can narrow our choice down even further, to a finite set of subedges of the simplification, and still sufficiently preserve the quality of the solution. In Section 3, we present our main algorithm. The algorithm uses the concepts and techniques developed in Section 2 in combination with the multiplicative weights update method. In Section 4, we analyze the approximation factor and running time of this algorithm. Crucially, we show that the VC-dimension of the induced set system which is implicitly used by our algorithm is small by design.

2 Structuring the solution space

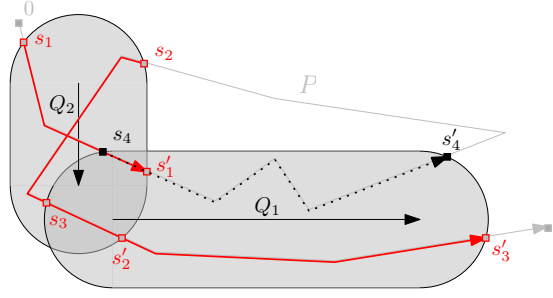
In this section, we introduce key concepts that allow us to transfer the problem to a set cover problem on a finite set system with small VC-dimension and still obtain good approximation bounds. The main result of this section is Theorem 14.

Simplifications and containers. We start by defining the notion of curve-simplification that we will use throughout the paper.

► **Definition 3 (simplification).** *Let P be a polygonal curve in \mathbb{R}^d . Let (t_1, \dots, t_n) be the vertex-parameters of P , and $p_i = P(t_i)$ the vertices of P . Consider an index set $1 \leq i_1 < \dots < i_k \leq n$ that defines vertices p_{i_j} . We call a curve S defined by such an ordered set of vertices $(p_{i_1}, \dots, p_{i_k}) \in (\mathbb{R}^d)^k$ a **simplification** of P . We say the simplification is **Δ -good**, if the following properties hold:*

- (i) $\|p_{i_j} - p_{i_{j+1}}\| \geq \frac{\Delta}{3}$ for $1 \leq j < k$
- (ii) $d_F(P[t_{i_j}, t_{i_{j+1}}], \overline{p_{i_j} p_{i_{j+1}}}) \leq 3\Delta$ for all $1 \leq j < k$.
- (iii) $d_F(P[t_1, t_{i_1}], \overline{p_{i_1} p_{i_1}}) \leq 3\Delta$ and $d_F(P[t_{i_k}, t_n], \overline{p_{i_k} p_{i_k}}) \leq 3\Delta$
- (iv) $d_F(P[t_{i_j}, t_{i_{j+2}}], \overline{p_{i_j} p_{i_{j+2}}}) > 2\Delta$ for all $1 \leq j < k - 1$

Our intuition is the following. Property (i) guarantees that S does not have short edges. Property (ii) and (iii) together tell us, that the simplification error is small. Property (iv) tells us, that the simplification is (approximately) maximally simplified, that is, we cannot remove a vertex, and hope to stay within Fréchet distance 2Δ to P .



■ **Figure 2** Example of the structured Δ -coverage of a set $C = \{Q_1, Q_2\}$ and a curve P . Here we have $\Psi'_\Delta(P, C) = [s_1, s'_1] \cup [s_2, s'_2]$ since the subcurves $P[s_1, s'_1]$ and $P[s_2, s'_2]$ have Fréchet distance Δ to Q_1 and $P[s_3, s'_3]$ has Fréchet distance Δ to Q_2 . Note that $[s_4, s'_4]$ is not part of the coverage since the subcurve $P[s_4, s'_4]$ consists of 4 edges.

► **Definition 4 (Container).** Let P be a polygonal curve, let $\pi = P[s, t]$ be a subcurve of P , and let (t_1, \dots, t_n) be the vertex-parameters of P . For a simplification S of P defined by index set $I = (i_1, \dots, i_k)$, define the **container** $c_S(\pi)$ of π on S as $S[t_a, t_b]$, with $a = \max(\{i_1\} \cup \{i \in I \mid t_i \leq s\})$ and $b = \min(\{i \in I \mid t_i \geq t\} \cup \{i_k\})$.

The following lemma has been proven by de Berg et al. [14]. We restate and reprove it here with respect to our notion of simplification.

► **Lemma 5 ([14]).** Let P be a polygonal curve in \mathbb{R}^d , and let S be a Δ -good simplification of P . Let Q be an edge in \mathbb{R}^d and let π be a subcurve of P with $d_F(Q, \pi) \leq \Delta$. Then $c_S(\pi)$ consists of at most 3 edges.

Proof. Assume for the sake of contradiction, that $c_S(\pi)$ contains 4 edges, that is it has three internal vertices s_1, s_2, s_3 . By Definition 4 these three vertices are also interior vertices of π . As the Fréchet distance $d_F(Q, \pi) \leq \Delta$, there are points $q_1, q_2, q_3 \in Q$, that get matched to s_1, s_2 and s_3 respectively during the traversal, with $\|s_i - q_i\| \leq \Delta$. This implies $d_F(\pi[s_1, s_3], \overline{q_1 q_3}) \leq \Delta$. It also implies, that $d_F(\overline{s_1 s_3}, \overline{q_1 q_3}) \leq \Delta$. But then

$$d_F(\overline{s_1 s_3}, P[s_1, s_3]) = d_F(\overline{s_1 s_3}, \pi[s_1, s_3]) \leq d_F(\overline{s_1 s_3}, \overline{q_1 q_3}) + d_F(\pi[s_1, s_3], \overline{q_1 q_3}) \leq 2\Delta,$$

contradicting the assumption that S is a Δ -good simplification. ◀

Structured coverage and candidate space. We want to make use of the property of Δ -good simplifications shown in Lemma 5. For this we adapt the notion of Δ -coverage from Section 1 as follows.

► **Definition 6.** Let S be a polygonal curve in \mathbb{R}^d . Let (t_1, \dots, t_n) be the vertex-parameters of S . Let $\ell \in \mathbb{N}$ and $\Delta \in \mathbb{R}$ be fixed parameters. Define the **structured Δ -coverage** of a set of center curves $C \subset \mathbb{X}_\ell^d$ as

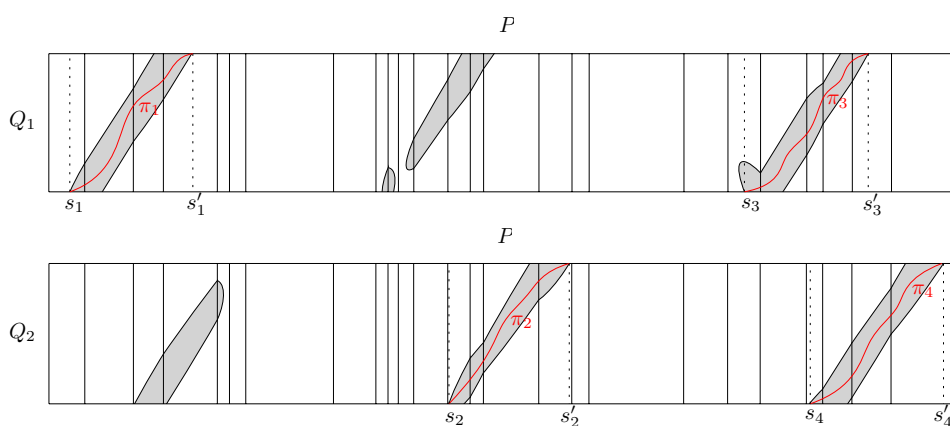
$$\Psi'_\Delta(S, C) = \bigcup_{q \in C} \bigcup_{(i,j) \in J} \Psi_\Delta^{(i,j)}(S, q)$$

where

$$\Psi_\Delta^{(i,j)}(S, q) = \{s \in [t, t'] \mid t_i \leq t \leq t_{i+1}; t \leq t'; t_{j-1} \leq t' \leq t_j; d_F(S[t, t'], q) \leq \Delta\},$$

and where $J = \{1 \leq i \leq j \leq n \mid 1 \leq j - i \leq 4\}$.

If it holds that $\Psi'_\Delta(S, C) = [0, 1]$, then we call C a **structured Δ -covering** of S .



■ **Figure 3** Free space diagrams of the curves P and Q_1 (resp. Q_2) depicted in Figure 1. The monotone paths π_i illustrate that the Fréchet distance between $P[s_i, s'_i]$ and Q_1 (resp. Q_2) is equal to Δ for $1 \leq i \leq 4$.

► **Observation 7.** *In general for any polygonal curve S and set of center curves C it holds that $\Psi'_\Delta(S, C) \subseteq \Psi_\Delta(S, C)$.*

We now want to restrict the candidate set to subedges of a simplification of the input curve, thereby imposing more structure on the solution space. For this we begin by defining a more structured parametrization of the set of edges of a polygonal curve.

► **Definition 8 (Edge space).** *We define the **edge space** $\mathbb{T}_n = \{1, \dots, n-1\} \times [0, 1]$. We denote the set of edges of P with $E(P)$.*

► **Definition 9 (Candidate space).** *Let $E = \{e_1, \dots, e_{n-1}\}$ be an ordered set of edges in \mathbb{R}^d . We define the **candidate space** induced by E as the set $\mathcal{Z}_E = \{(i_1, t_1, i_2, t_2) \in \mathbb{T}_n \times \mathbb{T}_n \mid i_1 = i_2\}$. We associate an element $(i, t_1, i, t_2) \in \mathcal{Z}_E$ with the subedge $e_i(t_1) e_i(t_2)$.*

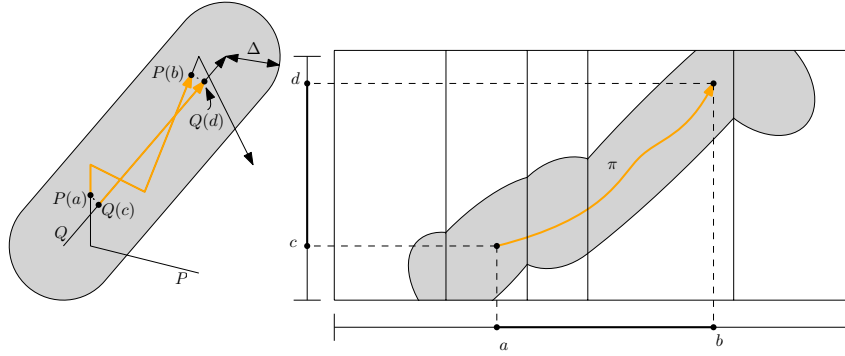
The following theorem summarizes and motivates the above definitions of structured coverage and candidate space. Namely, we can restrict the search space to subedges of the simplification S and still obtain a good covering of P . Moreover, we can evaluate the coverage of our solution solely based on S . The structured coverage only allows subcurves of S that consist of at most three edges to contribute to the coverage. This technical restriction is necessary to obtain a small VC-dimension in our main algorithm later on, and it is well-motivated by Lemma 5.

► **Theorem 10.** *Let S be a Δ -good simplification of a curve P . Let C be a set of subedges of edges of S . If C is a structured 8Δ -covering of S , then C is an 11Δ -covering of P . Moreover, if k is the size of an optimal Δ -covering of P , then there exists such a set C of size at most $3k$.*

Partial traversals and coverage. Our algorithm and analysis use the notion of the free space diagram which was first introduced by Alt and Godau [3]. It is instructive to consider this concept in the context of the coverage problem. Refer to Figure 3.

► **Definition 11 (Free space diagram).** *Let P and Q be two polygonal curves parametrized over $[0, 1]$. The free space diagram of P and Q is the joint parametric space $[0, 1]^2$ together with a not necessarily uniform grid, where each vertical line corresponds to a vertex of P and each horizontal line to a vertex of Q . The Δ -free space of P and Q is defined as*

$$\mathcal{D}_\Delta(P, Q) = \{(x, y) \in [0, 1]^2 \mid \|P(x) - Q(y)\| \leq \Delta\}$$



■ **Figure 4** An illustration of a Δ -feasible $(2, 4)$ -partial traversal π from (a, c) to (b, d) of P and Q . π covers all points between a and b on P , and all points between c and d on Q .

This is the set of points in the parametric space, whose corresponding points on P and Q are at a distance at most Δ . The edges of P and Q segment the free space into cells. We call the intersection of $\mathcal{D}_\Delta(P, Q)$ with the boundary of cells the **free space intervals**.

Alt and Godau [3] showed that the Δ -free space inside any cell is convex and has constant complexity. More precisely, it is an ellipse intersected with the cell. Furthermore, the Fréchet distance between two curves is less than or equal to Δ if and only if there exists a path $\pi : [0, 1] \rightarrow \mathcal{D}_\Delta(P, Q)$ that starts at $(0, 0)$, ends in $(1, 1)$ and is monotone in both coordinates.

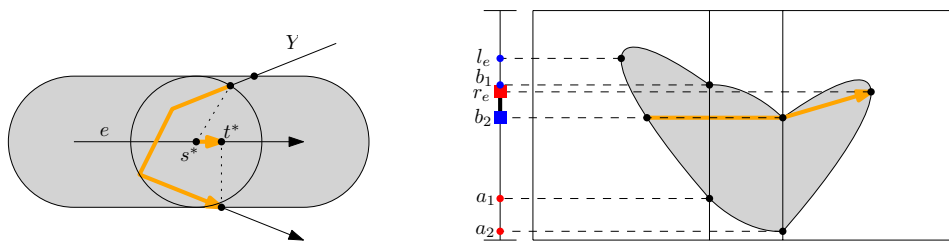
► **Definition 12** (Partial traversal). Let P be a polygonal curve in \mathbb{R}^d , and let (t_1, \dots, t_n) be the vertex-parameters of P . Let $1 \leq i < j \leq n$ be integer values. Let Q be an edge in \mathbb{R}^d . We define an (i, j) -**partial traversal** as a pair of continuous, monotone increasing and surjective functions, $f : [0, 1] \rightarrow [a, b]$ and $g : [0, 1] \rightarrow [c, d]$, where $t_i \leq a \leq t_{i+1}$, $t_{j-1} \leq b \leq t_j$, $0 \leq a \leq b \leq 1$, and $0 \leq c \leq d \leq 1$. We say that (f, g) is a partial traversal from (a, c) to (b, d) .

► **Definition 13** (Δ -feasible). We say that a partial traversal is **Δ -feasible** if the image of the path $\pi : [0, 1] \rightarrow [0, 1]^2$ defined by $\pi(t) = (f(t), g(t))$ is contained inside the Δ -free space $\mathcal{D}_\Delta(P, Q)$. We say that π **covers** a point t on P if $t \in [a, b]$ and we say that π covers a point t on Q if $t \in [c, d]$.

A finite set of candidates. By Theorem 10, it is sufficient to find a structured covering using a suitable simplification of the input curve. However the corresponding search space would still be infinite, even for a single edge. We will next define a finite set of candidates and show that it contains a good solution. In particular, our goal is to prove the following theorem.

► **Theorem 14.** Let P be a polygonal curve of complexity n in \mathbb{R}^d and let $\Delta > 0$ be given. Let S be a Δ -good simplification of P . Assume there exists a Δ -covering C of P of cardinality k . Then, there exists an algorithm that computes in $O(n^3)$ time and space a set of candidates $B \subset \mathcal{Z}_E(S) \subset \mathbb{X}_2^d$ with $|B| \in O(n^3)$, such that B contains a structured 8Δ -covering C_B of S of size at most $12k$. Moreover, C_B is a 11Δ -covering of P .

The main steps to constructing this set of candidates B are as follows. We first define a special set of subcurves of the simplification S . Intuitively, these are the containers of S of subcurves of P that may contribute to the coverage.



■ **Figure 5** Examples of extremal points. Shown on the right are the two free space intervals $[a_1, b_1]$ and $[a_2, b_2]$ as well as the left- and rightmost points l_e and r_e of the Δ -free space of e and Y . The extremal points are defined by b_2 and r_e . All points considered for the first extremal point are shown in blue. Similarly all points considered for the second extremal point are shown in red. A traversal from the first extremal point to the second extremal point is illustrated. On the left the resulting subedge $e[s^*, t^*]$ and the maximal subcurve of Y that can be matched are illustrated.

► **Definition 15** (Generating subcurves). Let S be a Δ -good simplification of a polygonal curve P . Let (t_1, \dots, t_m) be the vertex-parameters of S . For any $1 \leq i, 1 \leq j \leq 3$ and $i + j \leq m$, we say the subcurve $S[t_i, t_{i+j}]$ is a **generating subcurve**. In particular, this defines all subcurves of at most three edges starting and ending at vertices of S .

Now, for every generating subcurve Y of S and every edge e of S , we can identify an interval defining a subedge of e , that maximizes the Δ -coverage on Y over all subedges of e . For this reason, we call the endpoints of this interval extremal. Using this definition we define the finite candidate set induced by S via generating triples.

► **Definition 16** (Δ -extremal points). Given a value of $\Delta > 0$, a polygonal curve $Y : [0, 1] \rightarrow \mathbb{R}^d$ of m edges and an edge $e : [0, 1] \rightarrow \mathbb{R}^d$, such that they permit a Δ -feasible $(1, m)$ -partial traversal. As e is a single edge, the Δ -free space of Y and e consists of a single row. Let $[a_i, b_i]$ be the i th vertical free space interval of the Δ -free space of Y and e . Denote by $l = (l_Y, l_e)$ the leftmost point in the Δ -free space of Y and e and $r = (r_Y, r_e)$ the rightmost point (in case l is not unique, chose the point with smallest y -coordinate, and r as the point with the biggest y -coordinate). We define the **Δ -extremal points** induced by Y on e as the tuple $\mathcal{E}_\Delta(Y, e) = (s, t) \in [0, 1]^2$ with $s = \min(\{l_e\} \cup \{b_1, \dots, b_{n-1}\})$ and $t = \max(\{r_e\} \cup \{a_1, \dots, a_{n-1}\})$. We explicitly allow that $t < s$.

► **Definition 17** (Generating triples). Let S be a Δ -good simplification of a polygonal curve P . We define the set of **generating triples** T_S as a set of triples (e, Y_1, Y_2) , where e is any edge of S , and Y_1 and Y_2 are generating subcurves of S (not necessarily distinct). We include the triple (e, Y_1, Y_2) in the set T_S if and only if there are points $p \in e$, $p_1 \in Y_1$ and $p_2 \in Y_2$ such that $\|p - p_1\| \leq 8\Delta$ and $\|p - p_2\| \leq 8\Delta$.

► **Definition 18** (Candidate set). Let $\Delta > 0$ be a given value and let S be a Δ -good simplification of a polygonal curve P . Let T_S be the set of generating triples of S . We define the **candidate set** induced by S with respect to Δ as the set of line segments

$$B = \{e[s_1, t_2] \mid \exists (e, S_1, S_2) \in T_S, \text{ s.t. } \mathcal{E}_{8\Delta}(S_i, e) = (s_i, t_i) \text{ for } i \in \{1, 2\}\}$$

Clearly, the set B can be computed in $O(|T_S|)$ time and space, if the set T_S is given.

In the full version [7], we show that, for any suitable covering, we can deform each subedge of the solution to one of our candidates while retaining the coverage on a fixed subcurve. However, while retaining coverage on one subcurve, we may lose coverage on

another subcurve in the same cluster. We show, through a case analysis, how to deal with all subcurves at once while increasing the number of clusters by a factor of at most 4. We use this fact together with Theorem 10 to prove Theorem 14.

3 The main algorithm

We describe the main algorithm below with pseudocode specified in Algorithm 1 and Algorithm 2. Specifications of the missing subroutines are given in Table 1. Several additional building blocks of the algorithm are described in the full version [7]: computing candidates, computing the structured coverage, testing feasibility and computing simplifications.

Algorithm. The algorithm receives as input a polygonal curve P in \mathbb{R}^d and a parameter $\Delta \geq 0$. The goal is to compute a small set of edges C , such that all points on P are covered by the Δ' -coverage of C on P for some $\Delta' \in O(\Delta)$. The algorithm APPROXCOVER (see Algorithm 1), when called with input P and Δ , first computes a Δ -good simplification S of P and generates a finite subset B of the candidate space $\mathcal{Z}_{E(S)} \subset \mathbb{X}_2^d$ defined on the edges of this simplification. For this, we use the construction of the candidate set presented in Section 2. The algorithm then performs an exponential search with the variable k that controls the target size of the solution. Starting with a constant k , the algorithm tries to find a solution of size approximately k and if this fails, the algorithm doubles k and continues. For finding a solution with fixed target size, the algorithm KAPPROXCOVER is used (see Algorithm 1). This algorithm is called with the simplification S , the candidate set B and set of parameters r, Δ', k' , and i_{\max} . The algorithm KAPPROXCOVER uses a variant of the multiplicative weight update method with a maximum number of (proper) iterations bounded by i_{\max} . In the i th iteration, we take a sample from a discrete probability distribution \mathcal{D}_i that is defined on B via a weight function $w_i : B \rightarrow \mathbb{R}$, where the probability of an element $e \in B$ being in the sample is defined as $w_i(e) / \sum_{e \in B} w_i(e)$. For the initial distribution \mathcal{D}_1 , all weights are set to 1, which corresponds to the uniform distribution over B . During the course of the algorithm, we repeatedly update this distribution thereby generating distributions $\mathcal{D}_1, \mathcal{D}_2, \dots$ (up to $\mathcal{D}_{i_{\max}}$, unless the algorithm finds a solution in an earlier iteration). The update step performed by a call to subroutine UPDATEWEIGHT proceeds by doubling the weight of the subset F of B . This can be done in $O(|B|)$ time and space by storing the cumulative probability distribution.

With this basic mechanism in place, the algorithm KAPPROXCOVER now proceeds as follows. In each iteration, the algorithm computes a set $C \subset B$ by taking k' independent draws from the current distribution \mathcal{D}_i . Then, the algorithm checks, if C is a solution to our problem by a call to the subroutine POINTNOTCOVERED. The subroutine should either return that all points on S are in the Δ' -coverage of the solution C , or return a point t on S that is not covered in this way. This can be done by computing the structured coverage $\Psi'_{\Delta'}(S, C)$ explicitly. In the former case, the algorithm returns the solution and terminates. In the latter case, we compute the subset F of candidates B that would cover t with respect to the subcurves that contain t and which have at most 3 edges. To compute F , we simply iterate over all elements of B and check if t is covered by a call to ISFEASIBLE (see Algorithm 2). (For technical reasons, we parametrize the curve P via the edge space of the set of edges of P , so that we can locate the edge that contains t in constant time.) It is important that F is not a multiset, so repeated additions of an element will not increase its weight.

At this point we would like to perform the weight update step which we described above with respect to the set F , however, we only do this if the weight of the set F is small. If the total weight of the set F is larger than a $\frac{1}{r}$ -fraction of the total weight of B , then we simply skip the update step and continue by taking another sample from the current distribution.

■ **Algorithm 1** Main algorithm.

```

1: procedure APPROXCOVER( $P \in \mathbb{X}_n^d, \Delta \in \mathbb{R}$ )
2:    $S \leftarrow \text{SIMPLIFYCURVE}(P, \Delta)$ 
3:    $B \leftarrow \text{GENERATECANDIDATES}(S, \Delta)$ 
4:    $k \leftarrow 1$ 
5:    $\gamma \leftarrow 110d + 412$  ▷ bound on the VC-dimension
6:   repeat
7:      $k \leftarrow 2k$  ▷ increase target size for solution
8:      $r \leftarrow 2k, \Delta' \leftarrow \alpha\Delta, k' \leftarrow \lceil 16k\gamma \log(16k\gamma) \rceil, i_{\max} \leftarrow 5k \log_2(\frac{|B|}{k})$ 
9:      $C \leftarrow \text{KAPPROXCOVER}(S, B, r, \Delta', k', i_{\max})$  ▷ search solution with this size
10:  until  $C \neq \emptyset$  ▷ until we find a solution
11:  return  $C$ 

1: procedure KAPPROXCOVER( $S \in \mathbb{X}_n^d, B \subset \mathbb{X}_2^d, r, \Delta' \in \mathbb{R}, k', i_{\max} \in \mathbb{N}$ )
2:  Let  $\mathcal{D}_1$  be the uniform distribution over  $B$  with weight function  $w_1 : B \rightarrow \{1\}$ 
3:   $i \leftarrow 1$ 
4:  repeat
5:     $C \leftarrow$  sample  $k'$  elements from  $\mathcal{D}_i$ 
6:     $t \leftarrow \text{POINTNOTCOVERED}(C, S, \Delta')$ 
7:    if  $t = -1$  then return  $C$  ▷ if all points covered, return solution found
8:     $F \leftarrow \emptyset$  ▷ otherwise, compute feasible set of  $t$ 
9:    for each  $Q \in B$  do
10:     if  $\text{ISFEASIBLE}(Q, S, t, \Delta')$  then add  $Q$  to  $F$ 
11:     if  $\Pr_{\mathcal{D}_i}[F] \leq \frac{1}{r}$  then
12:        $\mathcal{D}_{i+1} \leftarrow \text{WEIGHTUPDATE}(\mathcal{D}_i, F)$  ▷ increase the probability of  $F$ 
13:        $i \leftarrow i + 1$ 
14:  until  $i > i_{\max}$ 
15:  return  $\emptyset$  ▷ no solution found for this target size

```

■ **Algorithm 2** Subroutine ISFEASIBLE which is called by the main algorithm.

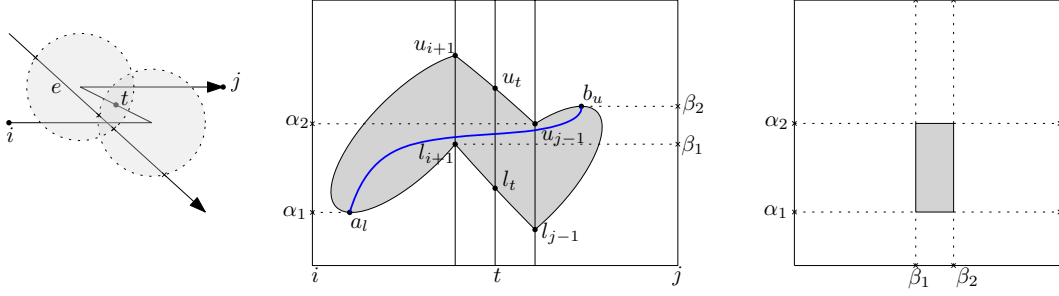
```

1: procedure ISFEASIBLE( $Q \in \mathbb{X}_2^d, S \in \mathbb{X}_n^d, t \in \mathbb{T}_n, \Delta' \in \mathbb{R}$ )
2:    $(t', i') \leftarrow t$  ▷ locate edge of  $t$  on  $S$ 
3:    $J = \{1 \leq i \leq j \leq n \mid 1 \leq j - i \leq 4; i \geq i' - 3; j \leq i' + 4\}$  ▷ find generating subcurves
4:   for  $(i, j) \in J$  do ▷ check if  $Q$  covers  $t$  on  $S$ 
5:     if  $t \in \Psi_{\Delta'}^{i,j}(S, Q)$  then return true
6:   return false

```

■ **Table 1** Specification of additional subroutines used in the main algorithm.

Procedure	Input	Output
SIMPLIFYCURVE	$P \in \mathbb{X}_n^d, \Delta \geq 0$	Δ -good simplification of P (Def. 3)
GENERATECANDIDATES	$S \in \mathbb{X}_n^d, \Delta \geq 0$	candidate set (Def. 18)
POINTNOTCOVERED	$C \subset \mathbb{X}_2^d, S \in \mathbb{X}_n^d, \Delta \geq 0$	either $t \in \mathbb{T}_n \setminus \Psi_{\Delta'}(S, C)$ or -1 if this set is empty
WEIGHTUPDATE	distribution \mathcal{D} given by weight function $w : B \rightarrow \mathbb{R}, F \subset B$	\mathcal{D}' with $w' : B \rightarrow \mathbb{R}$ where weight is doubled for all elements of F



■ **Figure 6** Example for the construction of the rectangle $R = [\alpha_1, \alpha_2] \times [\beta_1, \beta_2]$ for fixed P, i, j, t, Δ and e . The left image shows the curves $P[t_i, t_j]$ and e with two circles of radius Δ around $P(t_{i+1})$ and $P(t_{j-1})$. The middle image shows the corresponding Δ -free space diagram with a (i, j) -partial traversal from a_i to b_u and the right image shows the rectangle R in the parameter space $[0, 1]^2$ of e .

4 Analysis of the main algorithm

The algorithm described in Section 3 is based on the set cover algorithm by Brönniman and Goodrich [6]. A crucial step in the analysis of this algorithm is the analysis of the VC-dimension of the dual set system. In our case this is a set system formed by the sets F computed in the main algorithm. For the formal analysis of this set system, we introduce the notion of feasible sets.

► **Definition 19** (Feasible set). Let $S : \mathbb{T}_n \rightarrow \mathbb{R}^d$ be a polygonal curve and let $B \subset \mathbb{X}_2^d$ be a candidate set of edges and let $\Delta \geq 0$ be a real value. For any point $t \in \mathbb{T}_n$, we define the **feasible set** of t as the set of elements $Q \in B$ that admit an (i, j) -partial traversal with S that fully covers Q and that covers t on S , with the additional condition that $j - i \leq 3$. We denote the feasible set of t with $F_\Delta(t)$.

Note that for any fixed S and Δ the set of feasible sets $\{F_\Delta(t) \mid t \in \mathbb{T}_n\}$ is exactly the set system determined by the subroutine ISFEASIBLE described in Algorithm 2. We claim that any feasible set can be split into sets corresponding to the edges of the simplification, where each set consists of a constant union of rectangles in the candidate space restricted to the respective edge. Figure 6 illustrates one of those rectangles. The following lemma provides the formal statement.

► **Lemma 20.** Let P be a polygonal curve in \mathbb{R}^d and let $e \in \mathbb{X}_2^d$ be an edge. Let (t_1, \dots, t_n) be the vertex-parameters of P . For any integer values $1 \leq i < j \leq \min(i + 3, n)$ and real value $t \in [0, 1]$ with $t_i \leq t \leq t_j$, either there exist $\alpha_1, \alpha_2, \beta_1, \beta_2$ such that

$$R := \{(\alpha, \beta) \in [0, 1]^2 \mid t \in \Psi_\Delta^{i,j}(P, e[\alpha, \beta])\} = [\alpha_1, \alpha_2] \times [\beta_1, \beta_2],$$

or the set R is empty. Moreover, each α_v (respectively β_v) for $v \in \{1, 2\}$ can be written as $\alpha_v = c_v + \sqrt{d_v}$ (respectively $\beta_v = e_v + \sqrt{f_v}$), where the parameters c_v and d_v (respectively e_v and f_v) can be computed by an algorithm that takes $(i, j), t$ and e as input and needs $O(d)$ simple operations.

To prove a VC-dimension bound of $O(d)$, we combine the above lemma with the following general theorem which can be attributed to Goldberg and Jerrum [16]. We use the variant by Anthony and Bartlett [4], which is stated as follows.

► **Theorem 21** (Theorem 8.4 [4]). Suppose h is a function from $\mathbb{R}^a \times \mathbb{R}^b$ to $\{0, 1\}$ and let $H = \{x \rightarrow h(\alpha, x) \mid \alpha \in \mathbb{R}^a\}$ be the class determined by h . Suppose that h can be computed by an algorithm that takes as input the pair $(\alpha, x) \in \mathbb{R}^a \times \mathbb{R}^b$ and returns $h(\alpha, x)$ after no more than t simple operations. Then, the VC-dimension of H is $\leq 4a(t + 2)$.

► **Lemma 22.** *Let $S : \mathbb{T}_n \rightarrow \mathbb{R}^d$ be a polygonal curve and let $\Delta \in \mathbb{R}_+$. Consider the set system $\{F_\Delta(t) \mid t \in \mathbb{T}_n\}$ with ground set \mathbb{X}_2^d . The VC-dimension of this set system is in $O(d)$.*

Proof. Define a function $h : \mathbb{T}_n \times \mathbb{X}_2^d \rightarrow \{0, 1\}$ with $h(t, Q) = 1$ if and only if a call to $\text{ISFEASIBLE}(Q, S, t, \Delta)$ returns true. We analyse the VC-dimension of the class of functions determined by h :

$$H = \{x \rightarrow h(t, x) \mid t \in \mathbb{T}_n\}$$

As a consequence, we obtain the same bounds on the VC-dimension of the corresponding set system \mathcal{R} with ground set \mathbb{X}_2^d where a set $r_t \in \mathcal{R}$ is defined by a $t \in \mathbb{T}_n$ with

$$r_t = \{Q \in \mathbb{X}_2^d \mid h(t, Q) = 1\}$$

In order to show the lemma, we first argue that for any given $t \in \mathbb{T}_n$ and $Q \in \mathbb{X}_2^d$ the expression $h(t, Q)$ can be evaluated with $O(d)$ simple operations.

Let $(t', i') = t$ and recall the index set $J = \{(i, j) \mid i' - 3 \leq i \leq i' \leq j \leq i + 3\}$ as in the procedure ISFEASIBLE . Note that $|J| = 9$ and that J can be determined by $O(1)$ simple operations from i' . Note that ISFEASIBLE returns true if and only if $t \in \Psi_\Delta^{i,j}(S, Q)$ for some $(i, j) \in J$. So, for fixed (i, j) , consider the set

$$R = \{(\alpha, \beta) \in [0, 1]^2 \mid t \in \Psi_\Delta^{i,j}(S, Q[\alpha, \beta])\}$$

Lemma 20 implies that R is either empty or can be written as a rectangle $[\alpha_1, \alpha_2] \times [\beta_1, \beta_2]$. Note that $t \in \Psi_\Delta^{i,j}(S, Q)$ if and only if R is non-empty and $(0, 1) \in R$. By Lemma 20, this test can be performed using $O(d)$ simple operations. Thus, we can apply Theorem 21 and conclude that the VC-dimension of H is in $O(d)$. ◀

With proper bounds on the VC-dimension in place, we obtain the following main result. The proof is based on the well-known $\frac{1}{r}$ -net theorem by Haussler and Welzl [20], which provides a bound on the probability that our sample chosen in line 5 is a $\frac{1}{r}$ -net of the weighted set system, based on the VC-dimension of this set system. We use this to bound the expected number of iterations of the main loop in KAPPROXCOVER within our analysis of the multiplicative weights update algorithm.

► **Theorem 23.** *Given a polygonal curve $P \in \mathbb{X}_n^d$ and $\Delta \in \mathbb{R}_+$, there exists an algorithm that computes an (α, β) -approximate solution to the Δ -coverage problem on P with $\alpha = 11$ and $\beta = O(\log k^*)$, where k^* is the minimum size of a solution to the Δ -coverage problem on P . The algorithm needs in expectation $\tilde{O}((k^*)^2 n + k^* n^3)$ time and $\tilde{O}((k^*) n + n^3)$ space.*

In the full version [7], we show improved bounds for c -packed curves. The only modification to the algorithm is a more careful generation of the triples that generate the candidate set.

► **Theorem 24.** *Let $P \in \mathbb{X}_n^d$ be c -packed and $\Delta \in \mathbb{R}_+$. Let k^* be the minimum size of a solution to the Δ -coverage problem on P . There exists an algorithm that outputs an $(11, O(\log(k^*)))$ -approximate solution. The algorithm needs*

1. $\tilde{O}((k^*)^2 n + nc^2 k^*)$ expected time and $\tilde{O}(k^* n + nc^2)$ space in \mathbb{R}^2 ,
2. $\tilde{O}((k^*)^2 n + nc^2 k^* + n^2)$ expected time and $\tilde{O}(k^* n + nc^2)$ space in \mathbb{R}^d .

In the full version [7], we also show that the property of the feasible sets as testified by Lemma 20 can be exploited to implicitly update the weights of a much larger set of candidates chosen from a uniform grid in the candidate space, thereby circumventing the explicit computation of the candidates. This improves the overall dependency on the complexity of the input curve in the running time, when compared to the previous algorithm – at the cost of a logarithmic factor of the relative arc-length of the curve.

► **Theorem 25.** Let $P \in \mathbb{X}_n^d$ and $\Delta \in \mathbb{R}_+$. Let k^* be the minimum size of a solution to the Δ -coverage problem on P . Let further $\lambda(P)$ be the arc length of the curve P . There exists an algorithm that outputs a $(12, O(\log(k^*)))$ -approximate solution. The algorithm needs in expectation $O(nk^{*3}(\log^4(\frac{\lambda(P)}{\Delta}) + \log^3(\frac{n}{k^*}) + n \log^2(n)))$ time and $O(nk^* \log(\frac{n\lambda(P)}{\Delta k^*}))$ space.

5 Conclusions

With the algorithm variants presented in this paper, we can find bicriteria-approximate solutions to the Δ -coverage problem on a polygonal curve P . The new algorithms improve upon previously known algorithms for the Δ -coverage problem both in terms of known running time and space requirement bounds [2], as well as approximation factors. To the best of our knowledge, our candidate generation leads to the first strongly polynomial algorithm for subtrajectory clustering under the continuous Fréchet distance that does not depend on the relative arclength λ/Δ of the input curve or the spread of the coordinates. The running time is at most cubic in n , the number of vertices of the input curve (Theorem 23). In practice, we expect this to be lower as testified by our analysis for c -packed curves (Theorem 24). The work of Gudmundsson et. al. [17] suggest that in practice most curves are c -packed for a c that is considerably smaller than the complexity of the curve. However, it remains to be seen if this also holds for the typically long curves which appear as input in the subtrajectory clustering setting. We also present a variant of the algorithm with implicit weight updates which achieves a linear dependency on n (Theorem 25) and this holds in general, without any c -packedness assumption on the input.

There are several avenues for future research. We mention some of them here. An interesting question that remains open for now is whether the implicit weight update can be performed directly on the candidate set (Definition 18). For this, we need to develop a dynamic data structure that can maintain the distribution on this candidate set to perform updates with rectangles and to sample from it. Another future research direction is to improve the dependency of the approximation factor on the parameter that controls the complexity of the input curves. Currently, the dependency is linear, and we did not try to improve it, since our focus was on clustering with line segments. Another interesting question is, how the low complexity center curves obtained by our algorithm can be best connected to center curves of higher complexity or even a geometric graph while retaining the Δ -covering.

References

- 1 Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, Abhinandan Nath, Jiangwei Pan, and Erin Taylor. Subtrajectory clustering: Models and algorithms. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '18*, pages 75–87, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3196959.3196972.
- 2 Hugo A. Akitaya, Frederik Brünig, Erin Chambers, and Anne Driemel. Subtrajectory clustering: Finding set covers for set systems of subcurves, 2021. doi:10.48550/ARXIV.2103.06040.
- 3 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.*, 5:75–91, 1995. doi:10.1142/S0218195995000064.
- 4 Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999. doi:10.1017/CB09780511624216.
- 5 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012. doi:10.4086/toc.2012.v008a006.

- 6 Hervé Brönnimann and Michael T Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995. doi:10.1007/BF02570718.
- 7 Frederik Brüning, Jacobus Conradi, and Anne Driemel. Faster approximate covering of subcurves under the fréchet distance, 2022. doi:10.48550/ARXIV.2204.09949.
- 8 Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristan, Rodrigo I. Silveira, Frank Staals, and Carola Wenk. Clustering trajectories for map construction. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '17*, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3139958.3139964.
- 9 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Jorren Hendriks, Erfan Hosseini Sereshgi, Vera Sacristán, Rodrigo I. Silveira, Jorrick Sleijster, Frank Staals, and Carola Wenk. Improved map construction using subtrajectory clustering. In *LocalRec'20: Proceedings of the 4th ACM SIGSPATIAL Workshop on Location-Based Recommendations, Geosocial Networks, and Geoadvertising, LocalRec@SIGSPATIAL 2020, November 3, 2020, Seattle, WA, USA*, pages 5:1–5:4, 2020. doi:10.1145/3423334.3431451.
- 10 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *Int. J. Comput. Geom. Appl.*, 21(3):253–282, 2011. doi:10.1142/S0218195911003652.
- 11 Maike Buchin, Bernhard Kilgus, and Andrea Kölzsch. Group diagrams for representing trajectories. *International Journal of Geographical Information Science*, 34(12):2401–2433, 2020. doi:10.1080/13658816.2019.1684498.
- 12 Maike Buchin and Carola Wenk. Inferring movement patterns from geometric similarity. *J. Spatial Inf. Sci.*, 21(1):63–69, 2020. doi:10.5311/JOSIS.2020.21.724.
- 13 Kenneth L Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM (JACM)*, 42(2):488–499, 1995. doi:10.1145/201019.201036.
- 14 Mark de Berg, Atlas F. Cook, and Joachim Gudmundsson. Fast Fréchet queries. *Computational Geometry*, 46(6):747–755, 2013. doi:10.1016/j.comgeo.2012.11.006.
- 15 Andrew T Duchowski. A breadth-first survey of eye-tracking applications. *Behavior Research Methods, Instruments, & Computers*, 34(4):455–470, 2002. doi:10.3758/BF03195475.
- 16 Paul W. Goldberg and Mark R. Jerrum. Bounding the Vapnik-Chervonenkis dimension of concept classes parameterized by real numbers. *Machine Learning*, 18:131–148, 1995. doi:10.1007/BF00993408.
- 17 Joachim Gudmundsson, Yuan Sha, and Sampson Wong. Approximating the packedness of polygonal curves, 2020. doi:10.48550/ARXIV.2009.07789.
- 18 Joachim Gudmundsson and Nacho Valladares. A GPU approach to subtrajectory clustering using the fréchet distance. *IEEE Trans. Parallel Distributed Syst.*, 26(4):924–937, 2015. doi:10.1109/TPDS.2014.2317713.
- 19 Joachim Gudmundsson and Sampson Wong. Cubic upper and lower bounds for subtrajectory clustering under the continuous fréchet distance, 2021. doi:10.48550/ARXIV.2110.15554.
- 20 David Haussler and Emo Welzl. Epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987. doi:10.1007/BF02187876.
- 21 Kenneth Holmqvist, Marcus Nyström, Richard Andersson, Richard Dewhurst, Halszka Jarodzka, and Joost Van de Weijer. *Eye tracking: A comprehensive guide to methods and measures*. OUP Oxford, 2011. URL: <https://global.oup.com/academic/product/eye-tracking-9780199697083>.
- 22 Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE transactions on pattern analysis and machine intelligence*, 36(7):1325–1339, 2013. doi:10.1109/TPAMI.2013.248.

28:16 Faster Approximate Covering of Subcurves Under the Fréchet Distance

- 23 Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 593–604, 2007. doi:10.1145/1247480.1247546.
- 24 Sen Qiao, Y. Wang, and J. Li. Real-time human gesture grading based on OpenPose. *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–6, 2017. doi:10.1109/CISP-BMEI.2017.8301910.
- 25 Roniel S. De Sousa, Azzedine Boukerche, and Antonio A. F. Loureiro. Vehicle trajectory similarity: Models, methods, and applications. *ACM Comput. Surv.*, 53(5), September 2020. doi:10.1145/3406096.
- 26 Sheng Wang, Zhifeng Bao, J Shane Culpepper, and Gao Cong. A survey on trajectory data management, analytics, and learning. *ACM Computing Surveys (CSUR)*, 54(2):1–36, 2021. doi:10.1145/3440207.
- 27 Guan Yuan, Penghui Sun, Jie Zhao, Daxing Li, and Canwei Wang. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1):123–144, 2017. doi:10.1007/s10462-016-9477-7.

Efficient Fréchet Distance Queries for Segments

Maike Buchin ✉

Ruhr Universität Bochum, Germany

Ivor van der Hoog ✉

Utrecht University, The Netherlands

Tim Ophelders ✉

Utrecht University, The Netherlands

TU Eindhoven, The Netherlands

Lena Schlipf ✉

Universität Tübingen, Germany

Rodrigo I. Silveira ✉

Polytechnic University of Catalonia, Barcelona, Spain

Frank Staals ✉

Utrecht University, The Netherlands

Abstract

We study the problem of constructing a data structure that can store a two-dimensional polygonal curve P , such that for any query segment \overline{ab} one can efficiently compute the Fréchet distance between P and \overline{ab} . First we present a data structure of size $O(n \log n)$ that can compute the Fréchet distance between P and a horizontal query segment \overline{ab} in $O(\log n)$ time, where n is the number of vertices of P . In comparison to prior work, this significantly reduces the required space. We extend the type of queries allowed, as we allow a query to be a horizontal segment \overline{ab} together with two points $s, t \in P$ (not necessarily vertices), and ask for the Fréchet distance between \overline{ab} and the curve of P in between s and t . Using $O(n \log^2 n)$ storage, such queries take $O(\log^3 n)$ time, simplifying and significantly improving previous results. We then generalize our results to query segments of arbitrary orientation. We present an $O(nk^{3+\varepsilon} + n^2)$ size data structure, where $k \in [1, n]$ is a parameter the user can choose, and $\varepsilon > 0$ is an arbitrarily small constant, such that given any segment \overline{ab} and two points $s, t \in P$ we can compute the Fréchet distance between \overline{ab} and the curve of P in between s and t in $O((n/k) \log^2 n + \log^4 n)$ time. This is the first result that allows efficient exact Fréchet distance queries for arbitrarily oriented segments.

We also present two applications of our data structure. First, we show that our data structure allows us to compute a local δ -simplification (with respect to the Fréchet distance) of a polygonal curve in $O(n^{5/2+\varepsilon})$ time, improving a previous $O(n^3)$ time algorithm. Second, we show that we can efficiently find a translation of an arbitrary query segment \overline{ab} that minimizes the Fréchet distance with respect to a subcurve of P .

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Computational Geometry, Data Structures, Fréchet distance

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.29

Related Version An abstract with preliminary results to this paper was presented at EuroCG 2020 [10]. A full version with all omitted proofs is available on ArXiv [11].

Preliminary Version: http://www1.pub.informatik.uni-wuerzburg.de/eurocg2020/data/uploads/papers/eurocg20_paper_65.pdf [10]

Full Version: <https://arxiv.org/abs/2203.01794> [11]

Funding Research of Schlipf was supported by the Ministry of Science, Research and the Arts Baden-Württemberg (Germany).



© Maike Buchin, Ivor van der Hoog, Tim Ophelders, Lena Schlipf, Rodrigo I. Silveira, and Frank Staals; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 29; pp. 29:1–29:14

Leibniz International Proceedings in Informatics



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

Acknowledgements This work started at Dagstuhl workshop 19352, “Computation in Low-Dimensional Geometry and Topology.” We thank Dagstuhl, the organizers, and the other participants for a stimulating workshop.

1 Introduction

Comparing the shape of polygonal curves is an important task that arises in many contexts such as GIS applications [2, 9], protein classification [20], curve simplification [7], curve clustering [1] and even speech recognition [21]. Within computational geometry, there are two well studied distance measures for polygonal curves: the Hausdorff and the Fréchet distance. The Fréchet distance has proven particularly useful as it takes the course of the curves into account. However, the Fréchet distance between curves is costly to compute: as its computation requires roughly quadratic time [3, 8]. When a large number of Fréchet distance queries are required, we would like to have a data structure, a so-called *distance oracle*, to answer these queries more efficiently. This leads to a fundamental data structuring problem: preprocess a polygonal curve such that, given a query polygonal curve, their Fréchet distance can be computed efficiently (query curves are assumed to be comparatively small). It turns out that this problem is extremely challenging. Indeed, even though great efforts have been devoted to design data structures to answer Fréchet distance queries, there is still no distance oracle known that is able to answer *exact* queries for a general query curve.

To make progress on this important problem, in this work we focus on a more restrictive but fundamental setting: when the query curve is a single segment. The reasons to study this variant of the problem are twofold. On the one hand, it is a necessary step to solve the general problem. On the other hand, it is a setting that has its own applications. For example, in trajectory simplification, or when trying to find subtrajectories that are geometrically close to a given query segment (e.g. when computing shortcut-variants of the Fréchet distance [12], or in trajectory analysis [5] on sports data). A similar strategy of tackling segment queries has also been successfully applied in nearest neighbor queries with the Fréchet distance [4].

We study preprocessing a polygonal curve P to determine the *exact continuous* Fréchet distance between P and a query segment in sublinear time. Specifically, we study preprocessing a polygonal curve P of n vertices in the plane, such that given a query segment \overline{ab} , traversed from a to b , the Fréchet distance between P and \overline{ab} can be computed in sublinear time. Without preprocessing, this problem can be solved in $O(n \log n)$ time.

Related work. Data structures that support (approximate) nearest neighbor queries with respect to the Fréchet distance have received considerable attention throughout the years [4, 13, 15]. In these problems, the goal is typically to store a set of polygonal curves such that given a query curve and a query threshold Δ one can quickly report (or count) the curves that are (approximately) within (discrete) Fréchet distance Δ of the query curve. Some of these data structures even allow approximately counting the number of curves that have a subcurve within Fréchet distance Δ [5]. Highlighting its practical importance, the near neighbor problem using Fréchet distance was posed as ACM Sigspatial GIS Cup in 2017 [24].

Here, we want to compute the Fréchet distance of (part of) a curve to a low complexity *query* curve. For the *discrete* Fréchet distance, efficient $(1 + \varepsilon)$ -approximate distance oracles are known, even when P is given in an online fashion [14]. For the *continuous* Fréchet distance, Driemel and Har-Peled [12] present an $O(n\varepsilon^{-4} \log \varepsilon^{-1})$ size data structure that given a query segment \overline{ab} can compute a $(1 + \varepsilon)$ -approximation of the Fréchet distance between P and \overline{ab} in $O(\varepsilon^{-2} \log n \log \log n)$ time. Their approach extends to higher dimensions and low complexity polygonal query curves. However, in contrast to our solution, this approximation uses techniques that do not provide insight into the structure of the algorithmic problem.

Gudmundsson et al. [17] present an $O(n \log n)$ sized data structure that can *decide* if the Fréchet distance to \overline{ab} is smaller than a given value Δ in $O(\log^2 n)$ time (so with some parametric search approach one could consider computing the Fréchet distance itself). However, their result holds only when the length of \overline{ab} and all edges in P is relatively large compared to Δ . De Berg et al. [6] presented an $O(n^2)$ size data structure that does not have any restrictions on the length of the query segment or the edges of P . However, the orientation of the query segment is restricted to be horizontal. Queries are supported in $O(\log^2 n)$ time, and even queries asking for the Fréchet distance to a vertex-to-vertex subcurve are allowed (in that case, using $O(n^2 \log^2 n)$ space). Recently, Gudmundsson et al. [18] extended this result to allow the subcurve to start and end anywhere within P . Their data structure has size $O(n^2 \log^2 n)$ and queries take $O(\log^8 n)$ time. In their journal version, Gudmundsson et al. [19] directly apply the main result of a preliminary version of this paper [10] to immediately improve space usage of their data structure to $O(n^{3/2})$; their preprocessing time remains $O(n^2 \log^2 n)$. The current version of this paper significantly improves these results. Moreover, we present data structures that allow for arbitrarily oriented query segments.

Problem statement & our results. Let P be a polygonal curve in \mathbb{R}^2 with n vertices p_1, \dots, p_n . For ease of exposition, we assume that the vertices of P are in general position, i.e., all x - and y -coordinates are unique, no three points lie on a line, and no four points are cocircular. We consider P as a function mapping any time $t \in [0, 1]$ to a point $P(t)$ in the plane. Our ultimate goal is to store P such that we can quickly compute the *Fréchet distance* $\mathcal{D}_{\mathcal{F}}(P, Q)$ between P and a query curve Q . The Fréchet distance is defined as

$$\mathcal{D}_{\mathcal{F}}(P, Q) = \inf_{\alpha, \beta} \max_{t \in [0, 1]} \|P(\alpha(t)) - Q(\beta(t))\|,$$

where $\alpha, \beta: [0, 1] \rightarrow [0, 1]$ are nondecreasing surjections, also called reparameterizations of P and Q , respectively, and $\|p - q\|$ denotes the Euclidean distance between p and q .

In this work we focus on the case where Q is a single line segment \overline{ab} starting at a and ending at b . Note that P may self-intersect and \overline{ab} may intersect P . Our first main result deals with the case where \overline{ab} is horizontal:

► **Theorem 1.** *Let P be a polygonal curve in \mathbb{R}^2 with n vertices. There is an $O(n \log n)$ size data structure that can be built in $O(n \log^2 n)$ time such that given a horizontal query segment \overline{ab} it can report $\mathcal{D}_{\mathcal{F}}(P, \overline{ab})$ in $O(\log n)$ time.*

This significantly improves over the earlier result of de Berg et al. [6], as we reduce the required space and preprocessing time from quadratic to near linear. We simultaneously improve the query time from $O(\log^2 n)$ to $O(\log n)$. We further extend our results to allow queries against subcurves of P . Let s, t be two points on P , we use $P[s, t]$ to denote the subcurve of P from s to t . For horizontal query segments we then get:

► **Theorem 2.** *Let P be a polygonal curve in \mathbb{R}^2 with n vertices. There is an $O(n \log^2 n)$ size data structure that can be built in $O(n \log^2 n)$ time such that given a horizontal query segment \overline{ab} and two query points s and t on P it can report $\mathcal{D}_{\mathcal{F}}(P[s, t], \overline{ab})$ in $O(\log^3 n)$ time.*

De Berg et al. presented a data structure that could handle such queries in $O(\log^2 n)$ time (using $O(n^2 \log^2 n)$ space), provided that s and t were vertices of P . Compared to their data structure we thus again significantly improve the space usage, while allowing more general queries. The recently presented data structure of Gudmundsson et al. [18] does allow s and t

to lie on the interior of edges of P (and thus supports queries against arbitrary subcurves). Their original data structure uses $O(n^2 \log^2 n)$ space and allows for $O(\log^8 n)$ time queries. Compared to their result we use significantly less space, while also improving the query time.

Using the insights gained in this restricted setting, we then present the first data structure that allows exact Fréchet distance queries with arbitrarily oriented query segments in sublinear time. With near quadratic space we obtain a query time of $O(n^{2/3} \log^2 n)$. If we insist on logarithmic query time the space usage increases to $O(n^{4+\varepsilon})$. In particular, we present a data structure with the following time-space tradeoff. At only a small additional cost we can also support subcurve queries.

► **Theorem 3.** *Let P be a polygonal curve in \mathbb{R}^2 with n vertices, and let $k \in [1..n]$ be a parameter. There is an $O(nk^{3+\varepsilon} + n^2)$ size data structure that can be built in $O(nk^{3+\varepsilon} + n^2)$ time such that given an arbitrary query segment \overline{ab} it can report $\mathcal{D}_{\mathcal{F}}(P, \overline{ab})$ in $O((n/k) \log^2 n)$ time. In addition, given two query points s and t on P , it can report $\mathcal{D}_{\mathcal{F}}(P[s, t], \overline{ab})$ in $O((n/k) \log^2 n + \log^4 n)$ time.*

In Theorem 3 and throughout the rest of the paper $\varepsilon > 0$ is an arbitrarily small constant. In both Theorem 2 and Theorem 3 the query time can be made sensitive to the number of vertices $m = |P[s, t]|$ in the query subcurve $P[s, t]$. That is, we can get query times $O(\log^3 m)$ and $O((m/k) \log^2 m + \log^4 m)$, respectively.

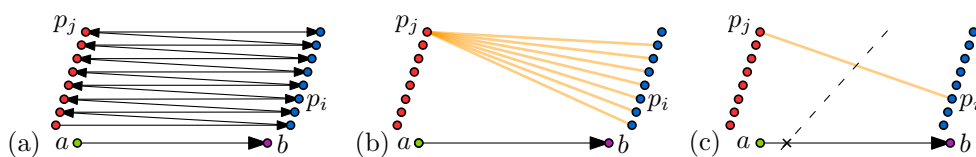
To achieve our results, we also develop data structures that allow us to efficiently query the directed Hausdorff distance $\overrightarrow{\mathcal{D}}_H(P[s, t], \overline{ab}) = \max_{p \in P[s, t]} \min_{q \in \overline{ab}} \|p - q\|$ from (a subcurve $P[s, t]$ of) P to the query segment \overline{ab} . For an arbitrarily oriented query segment \overline{ab} and a query subcurve $P[s, t]$ our data structure uses $O(n \log n)$ space and can answer such queries in $O(\log^2 n)$ time. Using more space, queries can be answered in $O(\log n)$ time, see Section 4.

Applications. In the full version [11] we show how to efficiently solve two problems using our data structure. First, we show how to compute a local δ -simplification of P – that is, a minimum complexity curve whose edges are within Fréchet distance δ to the corresponding subcurve of P – in $O(n^{5/2+\varepsilon})$ time. This improves existing $O(n^3)$ time algorithms [16].

Recently, Gudmundsson et al. [18] (full version [19]) studied the query version of this problem, where the goal is to preprocess P , such that given a query curve Q and two points s and t on P , one can find the translation of Q that minimizes the Fréchet distance between (a subcurve) $P[s, t]$ and Q efficiently. They study this query version in a restricted setting, where Q is a horizontal segment. Their original data structure uses $O(n^2 \log^2 n)$ space and allows for $O(\log^{32} n)$ time queries. By applying our data structure, we solve their problem using $O(n \log^2 n)$ space whilst supporting $O(\log^{12} n)$ time queries. In addition, we answer one of their open question to find optimal translations for arbitrarily oriented query segments. Finally, we answer another of their open problems by showing how to find a scaling of a query segment that minimizes the Fréchet distance. Specifically, we show a $O(n \log^2 n)$ size data structure that for any horizontal query segment, can compute the scaling of the segment that minimizes its Fréchet distance to $P[s, t]$ in $O(\log^4 n)$ time. We also show a version for arbitrarily oriented queries.

2 Global approach

We illustrate the main ideas of our approach, in particular for the case where the query segment \overline{ab} is horizontal, with a left of b . We can build a symmetric data structure in case a lies right of b . We now first review some definitions based on those in [6].



■ **Figure 1** (a) A polygonal curve and query segment. (b) The red vertex p_j forms a backward pair with all but one blue vertex. (c) For a fixed backward pair (p_i, p_j) , we consider the distance between the intersection (cross) of their bisector (dashed) and \overline{ab} , and either p_i or p_j .

Let P^{\leq} be the set of ordered pairs of vertices $(p, q) \in P \times P$ where p precedes or equals q along P . An ordered pair $(p, q) \in P^{\leq}$ forms a *backward pair* if $x_q \leq x_p$. Throughout the rest of the paper, x_p and y_p denote the x - and y -coordinates of point p , respectively. We denote by $\mathcal{B}(P)$ the backward pairs and say $(p, q) \in \mathcal{B}(P)$ is *trivial* if $p = q$ (Figure 1). For two points $p, q \in P$, we then define $\delta_{pq}(y) = \min_x \max \{ \| (x, y) - p \|, \| (x, y) - q \| \}$ (See Figure 1(c)). That is, $\delta_{pq}(y)$ is a function that for any y , gives the minimum distance between a point at height y and both p and q . We will use the function δ_{pq} only when $(p, q) \in \mathcal{B}(P)$ is a backward pair. We define the function $\mathcal{D}_B(y) = \max \{ \delta_{pq}(y) \mid (p, q) \in \mathcal{B}(P) \}$, which we refer to as the *backward pair distance* of a horizontal segment at height y with respect to P . Note that $\mathcal{D}_B(y)$ is the upper envelope of the functions δ_{pq} for all backward pairs (p, q) of P . De Berg et al. [6] prove that the Fréchet distance is the maximum of four terms:

$$\mathcal{D}_{\mathcal{F}}(P, \overline{ab}) = \max \left\{ \|p_1 - a\|, \|p_n - b\|, \overrightarrow{\mathcal{D}}_H(P, \overline{ab}), \mathcal{D}_B(y_a) \right\}. \quad (1)$$

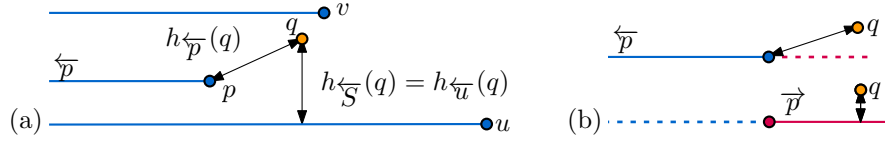
The first two terms are trivial to compute in $O(1)$ time. Like de Berg et al., we build separate data structures that allow us to efficiently compute the third and fourth terms.

A key insight is that we can compute $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$ by building the furthest segment Voronoi diagrams (FSVD) of two sets of horizontal halflines, and querying these diagrams with the endpoints a and b . See Section 3.1. This allows for a linear space data structure that supports querying $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$ in $O(\log n)$ time, improving both the space and query time over [6].

However, in [6] the data structure that supports computing the backward pair distance dominates the required space and preprocessing time, as there may be $\Omega(n^2)$ backward pairs, see Figure 1. Via a divide and conquer argument we show that the number of backward pairs that show up on the upper envelope \mathcal{D}_B is only $O(n \log n)$, see Section 3.2. The crucial ingredient is that there are only $O(n)$ backward pairs (p, q) contributing to \mathcal{D}_B in which p is a vertex among the first $n/2$ vertices of P , and q is a vertex in the remaining $n/2$ vertices. Surprisingly, we can again argue this using furthest segment Voronoi diagrams of sets of horizontal halflines. This allows us to build \mathcal{D}_B in $O(n \log^2 n)$ time in total. We can then extend these results to support queries against an arbitrary subcurve $P[s, t]$ of P (see [11]).

For arbitrarily oriented query segments we similarly decompose $\mathcal{D}_{\mathcal{F}}(P, \overline{ab})$ into four terms (Section 4). The directed Hausdorff term can still be queried efficiently using an $O(n \log^2 n)$ size data structure. However, our initial data structure for the backward pair distance uses $O(n^{4+\varepsilon})$ space. The main reason for this is that functions δ_{pq} expressing the cost of a backward pair are now bivariate, depending on both the slope and intercept of the supporting line of \overline{ab} . The upper envelope of a set of n such functions may have quadratic complexity.

While our divide and conquer strategy does not help us to directly bound the complexity of the (appropriately generalized function) \mathcal{D}_B in this case, it does allow us to support queries against subcurves of P . Moreover, we can use it to obtain a favourable query time vs. space trade off. In the full version [11] we then apply our data structure to efficiently solve various Fréchet distance related problems. Omitted proofs can be found in the full version [11].



■ **Figure 2** (a) A set $S := \{p, u, v\}$ with \overleftarrow{S} in blue. The distance $h_{\overleftarrow{p}}(q)$ is the distance to the apex of \overleftarrow{p} , whilst the distance $h_{\overleftarrow{u}}(q)$ is their vertical difference. (b) the distance between a point q and p is the maximum of the distance to the left and right halfline from p .

3 Horizontal queries

3.1 The Hausdorff term

In this section we introduce some definitions that will be used throughout the paper, and state the fact that there is a linear-size data structure to query the Hausdorff term in $O(\log n)$ time, which can be built in $O(n \log n)$ time. The proof of this result is in the full version [11].

For a point $p \in \mathbb{R}^2$, define \overleftarrow{p} to be the “leftward” horizontal halfline starting at p and containing all points directly to the left of p . Refer to Figure 2. Analogously, we define \overrightarrow{p} as the “rightward” horizontal halfline starting at p , so that $p = \overleftarrow{p} \cap \overrightarrow{p}$. We extend this notation to any set of points S , that is, $\overleftarrow{S} = \{\overleftarrow{s} \mid s \in S\}$ denotes the set of “leftward” halflines starting at the points in $S \subseteq \mathbb{R}^2$. We define \overrightarrow{S} analogously. Let S and T be two (possibly overlapping) point sets in the plane. We define the following distance functions for rays $\overleftarrow{p}, \overleftarrow{S}$ (definitions for $\overrightarrow{p}, \overrightarrow{S}$ are analogous):

$$h_{\overleftarrow{p}}(q) = \overrightarrow{\mathcal{D}}_H(\{q\}, \overleftarrow{p}) = \min \left\{ \|p' - q\| \mid p' \in \overleftarrow{p} \right\}, \quad h_{\overleftarrow{S}}(q) = \max \left\{ h_{\overleftarrow{p}}(q) \mid p \in S \right\}$$

Note that $h_{\overrightarrow{S}}$ (resp., $h_{\overleftarrow{S}}$) is the upper envelope of the distance functions to the halflines in \overrightarrow{S} (resp., \overleftarrow{S}). Since $h_{\overrightarrow{S}}$ and $h_{\overleftarrow{S}}$ map each point in the plane to a distance, the envelopes live in \mathbb{R}^3 . Combining furthest segment Voronoi diagrams with point location data structures, we can show how to compute $\overrightarrow{\mathcal{D}}_H(S, \overline{ab})$ efficiently:

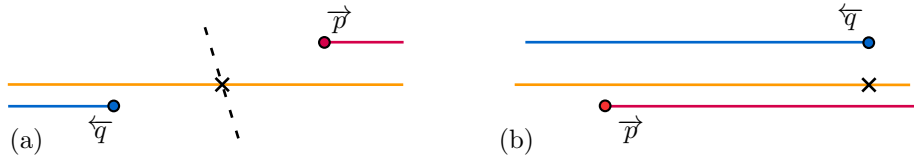
► **Theorem 4.** *Let S be a set of n points in \mathbb{R}^2 . In $O(n \log n)$ time we can build a data structure of linear size so that given a horizontal query segment \overline{ab} , $\overrightarrow{\mathcal{D}}_H(S, \overline{ab})$ can be computed in $O(\log n)$ time.*

Note that the directed Hausdorff distance from a polygonal curve P to a (horizontal) line segment is attained at a vertex of P [6], thus, we can use Theorem 4 to compute it.

3.2 The backward pairs term

Here we show that the function \mathcal{D}_B , representing the backward pair distance, has complexity $O(n \log n)$, can be computed in $O(n \log^2 n)$ time, and evaluated for some query value y in $O(\log n)$ time. This leads to an efficient data structure for querying P for the Fréchet distance to a horizontal query segment \overline{ab} , proving Theorem 1. See the full version [11] for details.

Recall that $\mathcal{D}_B(y)$ is the maximum over all function values $\delta_{pq}(y)$ for all backward pairs $(p, q) \in \mathcal{B}(P)$. To avoid computing $\mathcal{B}(P)$, we define a new function $\delta'_{pq}(y)$ that applies to any ordered pair of points $(p, q) \in P^{\leq}$. We show that for all backward pairs $(p, q) \in \mathcal{B}(P)$, we have $\delta'_{pq}(y) = \delta_{pq}(y)$. For any pair $(p, q) \in P^{\leq}$ that is not a backward pair, we show that there exists a (trivial) backward pair $(p', q') \in \mathcal{B}(P)$ such that $\delta'_{pq}(y) \leq \delta'_{p'q'}(y) = \delta_{p'q'}(y)$. Consequently, we can compute the value $\mathcal{D}_B(y)$ by computing the maximum value of $\delta'_{pq}(y)$ over all pairs in P^{\leq} . We will show how to do this in an efficient manner.



■ **Figure 3** The distance $\delta'_{pq}(y)$ is realised by either (a) the point of intersection between y and the bisector between \overleftarrow{q} and \overrightarrow{p} or, (b) the vertical distance between a line at height y and p or q .

For a pair of points $(p, q) \in P^{\leq}$, we define the *pair distance* between a query \overline{ab} at height y and (p, q) as the Hausdorff distance from a horizontal line of height y to $(\overrightarrow{p} \cup \overleftarrow{q})$, that is:

$$\delta'_{pq}(y) = \min_x \max \left\{ h_{\overleftarrow{q}}((x, y)), h_{\overrightarrow{p}}((x, y)) \right\}.$$

See Figure 3. The following key lemma states that, for our purposes we can use δ' instead of δ .

► **Lemma 5.** For any y , $\mathcal{D}_B(y) = \max \{ \delta_{pq}(y) \mid (p, q) \in \mathcal{B}(P) \} = \max \{ \delta'_{pq}(y) \mid (p, q) \in P^{\leq} \}$.

3.2.1 Relating $\mathcal{D}_B(y)$ to furthest segment Voronoi diagrams

Now we devise a divide and conquer algorithm that computes $\mathcal{D}_B(y)$ by computing it for subsets of vertices of P . Lemma 5 allows us to express $\mathcal{D}_B(y)$ in terms of P^{\leq} instead of $\mathcal{B}(P)$. Next we refine the definition of $\mathcal{D}_B(y)$ to make it decomposable. To that end, we define $\mathcal{D}_B(y)$ on pairs of subsets of P . Let S, T be any two subsets of vertices of P , we define:

$$\mathcal{D}_B^{S \times T}(y) = \max \{ \delta'_{pq}(y) \mid (p, q) \in (S \times T) \cap P^{\leq} \}.$$

We show that we can compute $\mathcal{D}_B^{S \times T}(y)$ efficiently using the δ' functions. For this, we fix a value of y and show that computing $\mathcal{D}_B^{S \times T}(y)$ is equivalent to computing an intersection between two curves that consist of a linear number of pieces, each of constant complexity. We then argue that as y changes, the intersection point moves along a linear complexity curve that can be computed in $O(n \log n)$ time. This will allow us to query $\mathcal{D}_B(y) = \mathcal{D}_B^{P \times P}(y)$ in $O(\log n)$ time, for any query height y .

From distance to intersections. For a fixed value y' , computing $\mathcal{D}_B^{S \times T}(y')$ is equivalent to computing an intersection point between two curves:

► **Lemma 6.** Let $y' \in \mathbb{R}$ be a fixed height, let p be a point in P , and let T be a subset of the vertices of $P[p, p_n]$. The graphs of the functions $x \mapsto h_{\overrightarrow{p}}((x, y'))$ and $x \mapsto h_{\overleftarrow{T}}((x, y'))$ intersect at a single point (x^*, y') . Moreover, $\mathcal{D}_B^{\{p\} \times T}(y') = h_{\overleftarrow{T}}((x^*, y')) = h_{\overrightarrow{p}}((x^*, y'))$

► **Lemma 7.** Let $y' \in \mathbb{R}$ be a fixed height, and let S, T be subsets of vertices of P such that all vertices in S precede all vertices in T . The graphs of the functions $x \mapsto h_{\overrightarrow{S}}((x, y'))$ and $x \mapsto h_{\overleftarrow{T}}((x, y'))$ intersect at a single point (x^*, y') . Moreover, $\mathcal{D}_B^{S \times T}(y') = h_{\overrightarrow{S}}((x^*, y')) = h_{\overleftarrow{T}}((x^*, y'))$.

Proof. If all points in S precede all points in T , then all elements in $S \times T$ are in P^{\leq} and we note: $\mathcal{D}_B^{S \times T}(y') = \max_{p \in S} \left\{ \mathcal{D}_B^{\{p\} \times T}(y') \right\}$. The equality then follows from Lemma 6. ◀

Given such a pair S, T , for a fixed value y' , we can compute a linear-size representation of $x \mapsto h_{\overleftarrow{T}}((x, y'))$ in $O(n \log n)$ time as follows (see Figure 4). We compute the *FSVD* of \overleftarrow{T} in $O(n \log n)$ time. Then, we compute the Voronoi cells intersected by a line of height y' (denoted by $\ell_{y'}$) in left-to-right order in $O(n \log n)$ time. Suppose that a segment of $\ell_{y'}$ intersects only the Voronoi cell belonging to a halfline $\overleftarrow{q} \in \overleftarrow{T}$, then on this domain the function $h_{\overleftarrow{T}}((x, y')) = h_{\overleftarrow{q}}((x, y'))$, and thus it has constant complexity. A horizontal line can intersect at most a linear number of Voronoi cells, hence the function has total linear complexity. Analogous arguments apply to $x \mapsto h_{\overrightarrow{S}}((x, y'))$.

Varying the y -coordinate. Let $f_{\overrightarrow{S}, \overleftarrow{T}} : y \mapsto x^*$ be the function that for each y gives the intersection point x^* such that $h_{\overrightarrow{S}}((x^*, y)) = h_{\overleftarrow{T}}((x^*, y))$. The intersection point (x^*, y) lies on a Voronoi edge of the *FSVD* of $(\overleftarrow{T} \cup \overrightarrow{S})$. More precisely, it lies on the bichromatic bisector of the *FSVD* of \overleftarrow{T} and the one of \overrightarrow{S} .

When we vary the y -coordinate, the intersection point traces this bisector. This implies that, given the *FSVD* of \overrightarrow{S} and the *FSVD* of \overleftarrow{T} , the graph of $f_{\overrightarrow{S}, \overleftarrow{T}}$ can be computed in $O(n)$ time. Using these properties, we can prove the following.

► **Lemma 8.** *Let S, T be subsets of vertices of P such that all vertices in S precede all vertices in T . The function $\mathcal{D}_B^{S \times T}$ has complexity $O(n)$ and can be computed in $O(n \log n)$ time. Evaluating $\mathcal{D}_B^{S \times T}(y)$, for some query value $y \in \mathbb{R}$, takes $O(\log n)$ time.*

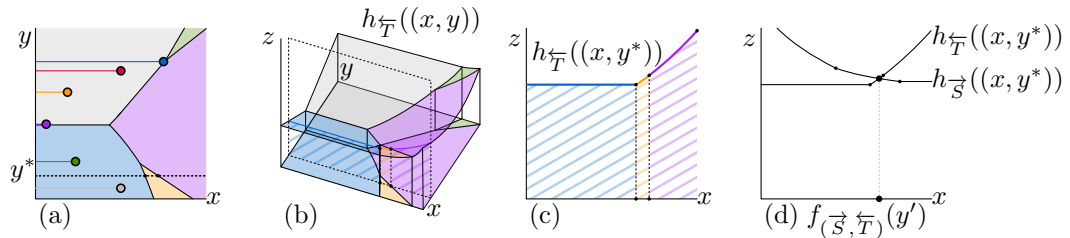
3.2.2 Applying divide and conquer

We begin by analyzing the complexity of the function $\mathcal{D}_B(y)$. Consider a partition of P into subcurves S and T with at most $\lceil n/2 \rceil$ vertices each, and with S occurring before T along P . Our approach relies on the following fact.

► **Observation 9.** *Let P be partitioned into two subcurves S and T with all vertices in S occurring on P before the vertices of T . We have that*

$$\mathcal{D}_B(y) = \mathcal{D}_B^{P \times P}(y) = \max \{ \mathcal{D}_B^{S \times S}(y), \mathcal{D}_B^{S \times T}(y), \mathcal{D}_B^{T \times T}(y) \}.$$

Note that we can omit $\mathcal{D}_B^{T \times S}$ because $(T \times S) \cap P^{\leq} = \emptyset$. We obtain the following lemma.



■ **Figure 4** (a) A set \overleftarrow{T} of rays arising from a set T of points, with their *FSVD*. (b) $h_{\overleftarrow{T}}((x, y))$ is the distance from (x, y) to the ray corresponding to the Voronoi cell at (x, y) . (c) For a fixed y' , $x \mapsto h_{\overleftarrow{T}}((x, y'))$ is monotonically increasing. (d) The value x for which $h_{\overleftarrow{T}}((x, y')) = h_{\overrightarrow{S}}((x, y'))$ corresponds to $f_{\overrightarrow{S}, \overleftarrow{T}}(y')$, and to $\mathcal{D}_B^{S \times T}(y')$.

► **Lemma 10.** *Let P be a polygonal curve with n vertices. The function \mathcal{D}_B has complexity $O(n \log n)$ and can be computed in $O(n \log^2 n)$ time. Evaluating $\mathcal{D}_B(y)$, for a given $y \in \mathbb{R}$, takes $O(\log n)$ time.*

Eq. 1 together with Theorem 4 and Lemma 10 thus imply that we can store P in an $O(n \log n)$ size data structure, so that we can compute $\mathcal{D}_{\mathcal{F}}(P, \overline{ab})$ for some horizontal query segment \overline{ab} in $O(\log n)$ time. That is, we have established Theorem 1.

Subcurve queries. We can extend our data structure to support Fréchet distance queries to subcurves of P , establishing Theorem 2. The two main ideas are (i) to store all intermediate data structures constructed in the above divide and conquer algorithm, and that (ii) we can actually achieve the result of Lemma 8 – evaluating $\mathcal{D}_B^{S \times T}(y')$ for some query value y' in $O(\log n)$ time – by *separately* storing a data structure on S and a data structure on T . See the full version [11] for details. Our data structure has total size $O(n \log^2 n)$ and allows us to find $O(\log n)$ nodes whose associated subcurves make up the query subcurve $P[s, t]$. For each such pair of nodes μ, ν we use the (extended) Lemma 8 data structures associated with these nodes to compute the contribution $\mathcal{D}_B^{P_\mu \times P_\nu}(y)$ of backward pairs with one vertex in subcurve P_ν and one vertex in P_μ . We thus spend $O(\log^3 n)$ time to compute the backward pair distance. This dominates the $O(\log^2 n)$ time required to query the Theorem 4 data structures associated with each node to compute the Hausdorff distance term.

4 Arbitrary orientation queries

In this section we extend our results to arbitrarily oriented query segments, proving Theorem 3. Let α be the slope of the line containing the query segment \overline{ab} , and let β be its intercept (note that vertical query segments can be handled by a rotated version of our data structure for horizontal queries). We again consider the case where a is left of b ; the other case is symmetric. Following Eq. 1, we can write $\mathcal{D}_{\mathcal{F}}(P, \overline{ab})$ as the maximum of four terms: $\|p_1 - a\|$, $\|p_n - b\|$, $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$, and the *backward pair distance* $\mathcal{D}_B(\alpha, \beta)$ with respect to α . The backward pair distance is now defined as

$$\begin{aligned} \mathcal{D}_B(\alpha, \beta) &= \max \{ \delta_{pq}(\alpha, \beta) \mid (p, q) \in \mathcal{B}(P) \}, & \text{where} \\ \delta_{pq}(\alpha, \beta) &= \min_x \max \{ \| (x, \alpha x + \beta) - p \|^2, \| (x, \alpha x + \beta) - q \|^2 \}. \end{aligned}$$

In Section 4.1 we present an $O(n \log n)$ size data structure that supports querying $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$ in $O(\log^2 n)$ time. The key insight is that we can use furthest *point* Voronoi diagrams instead of furthest *segment* Voronoi diagrams. In Section 4.2 we present a data structure that efficiently supports querying $\mathcal{D}_B(\alpha, \beta)$. In Section 4.3 we extend our results to support queries against subcurves of P as well. This combines our insights from the horizontal queries with our results from Sections 4.1 and 4.2. Finally, in Section 4.4 we then show how this also leads to a space-time trade off.

4.1 The Hausdorff distance term

For any point p and slope α we will denote by \overleftarrow{p}^α the ray with apex p and slope α that points in the leftward direction. Similarly, for any point set T , we define $\overleftarrow{T}^\alpha = \{ \overleftarrow{p}^\alpha \mid p \in T \}$. Furthermore, we define $h_{\overleftarrow{p}^\alpha}(x, y)$ to be the directed Hausdorff distance from (x, y) to the ray \overleftarrow{p}^α , and $h_{\overleftarrow{T}^\alpha}(x, y) = \max \{ h_{\overleftarrow{p}^\alpha}(x, y) \mid p \in T \}$.

29:10 Efficient Fréchet Distance Queries for Segments

We can show that we can use the furthest point Voronoi diagram instead of the furthest segment Voronoi diagram and obtain the following results.

► **Lemma 11.** *Let T be a set of n points in \mathbb{R}^2 . In $O(n^2)$ time we can construct an $O(n^2)$ size data structure so that given a query point (x, y) and slope α we can compute $h_{T, \alpha}^{\leftarrow}(x, y)$ in $O(\log n)$ time.*

► **Theorem 12.** *Let P be a polygonal curve in \mathbb{R}^2 with n vertices.*

■ *In $O(n \log n)$ time we can construct a data structure of size $O(n \log n)$ so that given a query segment \overline{ab} , $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$ can be computed in $O(\log^2 n)$ time.*

■ *In $O(n^2)$ time we can construct a data structure of size $O(n^2)$ so that given a query segment \overline{ab} , $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$ can be computed in $O(\log n)$ time.*

4.2 The backward pair distance term

Let $(p, q) \in P^{\leq}$ be an ordered pair. We restrict $\delta_{pq}(\alpha, \beta)$ to the interval of α values for which (p, q) is a backward pair with respect to orientation α . Hence, each δ_{pq} is a partially defined, constant algebraic degree, constant complexity, bivariate function. The backward pair distance \mathcal{D}_B is the upper envelope of $O(n^2)$ such functions. This envelope has complexity $O(n^{4+\varepsilon})$, for some arbitrarily small $\varepsilon > 0$, and can be computed in $O(n^{4+\varepsilon})$ time [23]. Evaluating $\mathcal{D}_B(\alpha, \beta)$ for some given α, β takes $O(\log n)$ time. The following lemma, together with Theorem 12 then gives an $O(n^{4+\varepsilon})$ size data structure that supports $O(\log n)$ time Fréchet distance queries.

► **Lemma 13.** *Let P be an n -vertex polygonal curve in \mathbb{R}^2 . In $O(n^{4+\varepsilon})$ time we can construct a size $O(n^{4+\varepsilon})$ data structure so that given a query segment \overline{ab} , $\mathcal{D}_B(\overline{ab})$ can be computed in $O(\log n)$ time.*

4.3 Subcurve queries

Next, we sketch how to support querying against subcurves $P[s, t]$ of P in $O(\log^4 n)$ time as well. Refer to the full version [11] for details. We use the same approach as in the horizontal query segment case: we store the vertices of P into the leaves of a range tree where each internal node ν corresponds to some canonical subcurve P_ν , so that any subcurve $P[s, t]$ can be represented by $O(\log n)$ nodes.

The Hausdorff distance term. Computing the directed Hausdorff distance is decomposable, so using this approach with the data structure of Theorem 12 immediately gives us a data structure that allows us to compute $\overrightarrow{\mathcal{D}}_H(P[s, t], \overline{ab})$ in $O(\log^2 n)$ time. Since the space usage satisfies the recurrence $S(n) = 2S(n/2) + O(n^2)$, this uses $O(n^2)$ space in total.

The backward pair distance term. By storing the data structure of Lemma 13 at every node of the tree, we can efficiently compute the contribution of the backward pairs inside each of the $O(\log n)$ canonical subcurves that make up $P[s, t]$. However, as before, we are still missing the contribution of the backward pairs from different canonical subcurves. We again store additional data structures at every node of the tree that allow us to efficiently compute this contribution.

Let S and T be (the vertices of) two such canonical subcurves, with all vertices of S occurring before T along P . As before we will argue that for some given α and β the functions $x \mapsto h_{T, \alpha}^{\leftarrow}(x, \alpha x + \beta)$ and $x \mapsto h_{S, \alpha}^{\rightarrow}(x, \alpha x + \beta)$ are monotonically increasing and decreasing,

respectively, and that the intersection point of (the graphs of) these functions corresponds to the contribution of the backward pairs in $S \times T$. So, our goal is to build data structures storing S and T that given a query α, β allow us to compute the intersection point of these functions. We will show that we can use the data structure of Lemma 11 to support such queries in $O(\log^2 n)$ time.

We generalize some of our earlier geometric observations to arbitrary orientations. Let p, q be vertices of P , and let S and T be subsets of vertices of P . We define

$$\delta'_{pq}(\alpha, \beta) = \min_x \max \left\{ h_{\leftarrow q}^\alpha((x, \alpha x + \beta)), h_{\rightarrow p}^\alpha((x, \alpha x + \beta)) \right\} \quad \text{and}$$

$$\mathcal{D}_B^{S \times T}(\alpha, \beta) = \max \left\{ \delta'_{pq}(\alpha, \beta) \mid (p, q) \in (S \times T) \cap P^\leq \right\}.$$

and prove the following lemmas (by essentially rotating the plane so that the query segment becomes horizontal, and applying the appropriate lemmas from earlier sections):

► **Lemma 14.** *Let P be partitioned into two subcurves S and T with all vertices in S occurring on P before the vertices of T . We have that*

$$\mathcal{D}_B(\alpha, \beta) = \mathcal{D}_B^{P \times P}(\alpha, \beta) = \max \left\{ \mathcal{D}_B^{S \times S}(\alpha, \beta), \mathcal{D}_B^{T \times T}(\alpha, \beta), \mathcal{D}_B^{S \times T}(\alpha, \beta) \right\}.$$

► **Lemma 15.** *Let S and T be subsets of vertices of P , with S occurring before T along P , and let α, β denote some query parameters. The function $x \mapsto h_{\leftarrow T}^\alpha(x, \alpha x + \beta)$ is monotonically increasing, whereas $x \mapsto h_{\rightarrow S}^\alpha(x, \alpha x + \beta)$ is monotonically decreasing. These functions intersect at a point $(x^*, \alpha x^* + \beta)$, for which $\mathcal{D}_B^{S \times T}(\alpha, \beta) = h_{\leftarrow T}^\alpha(x^*, \alpha x^* + \beta) = h_{\rightarrow S}^\alpha(x^*, \alpha x^* + \beta)$.*

Querying $\mathcal{D}_B^{S \times T}(\alpha, \beta)$. Consider the predicate $Q(x) = h_{\leftarrow T}^\alpha(x, \alpha x + \beta) < h_{\rightarrow S}^\alpha(x, \alpha x + \beta)$. It follows from Lemma 15 that there is a single value x^* so that $Q(x) = \text{FALSE}$ for all $x < x^*$ and $Q(x) = \text{TRUE}$ for all $x \geq x^*$. Moreover, x^* realizes $\mathcal{D}_B^{S \times T}(\alpha, \beta)$. By storing S and T , each in a separate copy of the data structure of Lemma 11, we can evaluate $Q(x)$, for any value x , in $O(\log n)$ time. We then use parametric search [22] to find x^* in $O(\log^2 n)$ time. Note that this approach is an $O(\log n)$ factor slower compared to the approach we used for horizontal queries only.

► **Lemma 16.** *Let S, T be subsets of vertices of P such that all vertices in S precede all vertices in T , stored in the data structure of Lemma 11. For any query α, β we can compute $\mathcal{D}_B^{S \times T}(\alpha, \beta)$ in $O(\log^2 n)$ time.*

For every node ν of the recursion tree on P we store: (i) the data structure of Lemma 13 built on its canonical subcurve P_ν , and (ii) the data structure of Lemma 11 built on the vertices of P_ν . The total space usage of the data structure follows the recurrence $S(n) = 2S(n/2) + O(n^{4+\varepsilon})$, which solves to $O(n^{4+\varepsilon})$. To query the data structure with some subcurve $P[s, t]$ from some vertex s to a vertex t we again find the $O(\log n)$ nodes whose canonical subcurves together define $P[s, t]$, query the Lemma 13 data structure for each of them, and run the algorithm from Lemma 16 for each pair. The total running time is then $O(\log^4 n)$. As before, the procedure can be easily extended to the case where s and t lie on the interior of an edge. We conclude:

► **Lemma 17.** *Let P be a polygonal curve in \mathbb{R}^2 with n vertices, and $\varepsilon > 0$. There is an $O(n^{4+\varepsilon})$ size data structure that can be built in $O(n^{4+\varepsilon})$ time that for an arbitrary query segment \overline{ab} (and two points s and t on P) can report $\mathcal{D}_B^{P[s, t] \times P[s, t]}(\alpha, \beta)$ in $O(\log^4 n)$ time.*

Since we can compute all four terms $\|s-a\|$, $\|t-b\|$, $\vec{\mathcal{D}}_H(P[s, t], \overline{ab})$, and $\mathcal{D}_B^{P[s, t] \times P[s, t]}(\alpha, \beta)$ in $O(\log^4 n)$ time, it follows that we can efficiently answer Fréchet distance queries against subcurves.

4.4 Space vs. Query time tradeoff

We can use our approach for subcurve queries from Section 4.3 to obtain a space vs. query time trade off for queries against the entire curve. Let $k \in [1..n]$ be a parameter. We trim the recursion tree on P at a node ν of size $O(k)$. Let \mathcal{T} denote the resulting tree (i.e., the top $\log(n/k)$ levels of the full recursion tree), and let $L(\mathcal{T})$ denote the set of leaves of \mathcal{T} , each of which thus corresponds to a subcurve of length $O(k)$. Let $\ell(\nu)$ and $r(\nu)$ be the left and right child of ν , respectively. By repeated application of the second equality in Lemma 14 we have

$$\mathcal{D}_B^{P \times P}(\alpha, \beta) = \max \left\{ \max_{\nu \in \mathcal{T}} \mathcal{D}_B^{P_{\ell(\nu)} \times P_{r(\nu)}}(\alpha, \beta), \max_{\nu \in L(\mathcal{T})} \mathcal{D}_B^{P_\nu \times P_\nu}(\alpha, \beta) \right\}.$$

At every leaf of \mathcal{T} we now store the data structure of Lemma 13, and at every internal node the data structure of Lemma 11. The space required by all Lemma 13 data structures is $O((n/k)k^{4+\varepsilon}) = O(nk^{3+\varepsilon})$. The total size for all Lemma 11 data structures follows the recurrence $S(n) = 2S(n/2) + O(n^2)$ which solves to $O(n^2)$. Hence, the total space used is $O(nk^{3+\varepsilon} + n^2)$. The preprocessing time is $O(nk^{3+\varepsilon} + n^2)$ as well.

To answer a query (α, β) we query the Lemma 13 data structures at the leaves of \mathcal{T} in $O(\log k)$ time each. For every internal node ν we use Lemma 16 to compute the contribution of $\mathcal{D}_B^{P_{\ell(\nu)} \times P_{r(\nu)}}(\alpha, \beta)$ in $O(\log^2 n)$ time. Hence, the total query time is $O((n/k) \log k + (n/k) \log^2 n) = O((n/k) \log^2 n)$. So, e.g., choosing $k = n^{1/3}$ yields an $O(n^{2+\varepsilon})$ size data structure supporting $O(n^{2/3} \log^2 n)$ time queries. We can extend this idea to support subcurve queries in $O((n/k) \log^2 n + \log^4 n)$ time as well, giving us the following result:

► **Lemma 18.** *Let P be a polygonal curve in \mathbb{R}^2 with n vertices, and let $k \in [1..n]$ be a parameter. In $O(nk^{3+\varepsilon} + n^2)$ time we can construct a data structure of size $O(nk^{3+\varepsilon} + n^2)$ so that given a query segment \overline{ab} , $\mathcal{D}_B(\overline{ab})$ can be computed in $O((n/k) \log^2 n)$ time. If, in addition we are also given two points s and t on P , $\mathcal{D}_B^{P[s,t] \times P[s,t]}(\overline{ab})$ can be computed in $O((n/k) \log^2 n + \log^4 n)$ time.*

Since computing $\overrightarrow{\mathcal{D}}_H(P[s,t], \overline{ab})$ can be done in $O(\log^2 n)$ time using only $O(n^2)$ space, we thus established Theorem 3. Once again, it is possible to make the query time proportional to the complexity of $P[s,t]$ rather than to n .

5 Concluding Remarks

We presented data structures for efficiently computing the Fréchet distance of (part of) a curve to a query segment. This constitutes an important step towards the more ambitious goal of finding data structures to efficiently answer queries for general polygonal curves.

Our results improve over previous work for horizontal segments and are the first for arbitrarily oriented segments. However, we are left with the challenge of reducing the space used for arbitrary orientations. There are two main issues. The first issue is that even for a small interval of query orientations (e.g., one of the $O(n^2)$ angular intervals defined by lines through a pair of points) it is difficult to limit the number of relevant backward pairs to $o(n^2)$. The second issue is how to combine the backward pair distance values contributed by various subcurves. For (low algebraic degree) univariate functions, the upper envelope has near linear complexity, whereas for bivariate functions the complexity is near quadratic. The combination of these issues makes it hard to improve over the somewhat straightforward $O(n^{4+\varepsilon})$ space bound we build upon.

References

- 1 Pankaj K Agarwal, Sariel Har-Peled, Nabil H Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005.
- 2 H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003.
- 3 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995.
- 4 Boris Aronov, Omrit Filtser, Michael Horton, Matthew J. Katz, and Khadijeh Sheikhan. Efficient nearest-neighbor query and clustering of planar curves. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R Salavatipour, editors, *Algorithms and Data Structures*, pages 28–42. Springer International Publishing, 2019.
- 5 M. de Berg, A. F Cook IV, and J. Gudmundsson. Fast Fréchet queries. *Computational Geometry*, 46(6):747–755, 2013.
- 6 Mark de Berg, Ali D Mehrabi, and Tim Ophelders. Data structures for Fréchet queries in trajectory data. In *29th Canadian Conference on Computational Geometry (CCCG'17)*, pages 214–219, 2017.
- 7 K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry & Applications*, 21(03):253–282, 2011.
- 8 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog: Improved bounds for computing the fréchet distance. *Discret. Comput. Geom.*, 58(1):180–216, 2017. doi:10.1007/s00454-017-9878-7.
- 9 Kevin Buchin, Maike Buchin, Marc van Kreveld, Maarten Löffler, Rodrigo I Silveira, Carola Wenk, and Lionov Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013.
- 10 Maike Buchin, Ivor van der Hoog, Tim Ophelders, Rodrigo I. Silveira, Lena Schlipf, and Frank Staals. Improved space bounds for Fréchet distance queries. In *36th European Workshop on Computational Geometry (EuroCG'20)*, 2020.
- 11 Maike Buchin, Ivor van der Hoog, Tim Ophelders, Rodrigo I. Silveira, Lena Schlipf, and Frank Staals. Efficient Fréchet distance queries for segments. *CoRR*, abs/2203.01794, 2022. arXiv:2203.01794.
- 12 A. Driemel and S. Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.
- 13 Anne Driemel and Ioannis Psarros. $(2 + \epsilon)$ -ANN for time series under the Fréchet distance. *arXiv preprint*, 2020. arXiv:2008.09406.
- 14 Arnold Filtser and Omrit Filtser. Static and streaming data structures for Fréchet distance queries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1150–1170. SIAM, 2021.
- 15 Arnold Filtser, Omrit Filtser, and Matthew J. Katz. Approximate nearest neighbor for curves – simple, efficient, and deterministic. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:19, 2020. doi:10.4230/LIPIcs.ICALP.2020.48.
- 16 Michael Godau. A natural metric for curves — computing the distance for polygonal chains and approximation algorithms. In *8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91)*, pages 127–136, 1991.
- 17 J. Gudmundsson, M. Mirzanezhad, A. Mohades, and C. Wenk. Fast Fréchet distance between curves with long edges. *International Journal of Computational Geometry & Applications*, 29(2):161–187, 2019.
- 18 Joachim Gudmundsson, André van Renssen, Zeinab Saeidi, and Sampson Wong. Fréchet distance queries in trajectory data. In *The Third Iranian Conference on Computational Geometry (ICCG 2020)*, pages 29–32, 2020.

29:14 Efficient Fréchet Distance Queries for Segments

- 19 Joachim Gudmundsson, André van Renssen, Zeinab Saeidi, and Sampson Wong. Translation invariant Fréchet distance queries. *Algorithmica*, 83(11):3514–3533, 2021. doi:10.1007/s00453-021-00865-0.
- 20 M. Jiang, Y. Xu, and B. Zhu. Protein structure–structure alignment with discrete Fréchet distance. *Journal of Bioinformatics and Computational Biology*, 6(01):51–64, 2008.
- 21 S. Kwong, QH He, K. Man, KS Tang, and CW Chau. Parallel genetic-based hybrid pattern matching algorithm for isolated word recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 12(05):573–594, 1998.
- 22 N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.
- 23 Micha Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete & Computational Geometry*, 12(3):327–345, 1994.
- 24 Martin Werner and Dev Oliver. ACM SIGSPATIAL GIS cup 2017: Range queries under Fréchet distance. *SIGSPATIAL Special*, 10(1):24–27, June 2018. doi:10.1145/3231541.3231549.

Search-Space Reduction via Essential Vertices

Benjamin Merlin Bumpus  

Eindhoven University of Technology, The Netherlands

Bart M. P. Jansen  

Eindhoven University of Technology, The Netherlands

Jari J. H. de Kroon  

Eindhoven University of Technology, The Netherlands

Abstract

We investigate preprocessing for vertex-subset problems on graphs. While the notion of kernelization, originating in parameterized complexity theory, is a formalization of provably effective preprocessing aimed at reducing the total instance size, our focus is on finding a non-empty vertex set that belongs to an optimal solution. This decreases the size of the remaining part of the solution which still has to be found, and therefore shrinks the search space of fixed-parameter tractable algorithms for parameterizations based on the solution size. We introduce the notion of a c -essential vertex as one that is contained in all c -approximate solutions. For several classic combinatorial problems such as ODD CYCLE TRANSVERSAL and DIRECTED FEEDBACK VERTEX SET, we show that under mild conditions a polynomial-time preprocessing algorithm can find a subset of an optimal solution that contains all 2-essential vertices, by exploiting packing/covering duality. This leads to FPT algorithms to solve these problems where the exponential term in the running time depends only on the number of *non-essential* vertices in the solution.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Packing and covering problems; Theory of computation → Linear programming; Theory of computation → Fixed parameter tractability

Keywords and phrases fixed-parameter tractability, essential vertices, covering versus packing

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.30

Related Version *Full Version*: <https://arxiv.org/abs/2207.00386> [7]

Funding Funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).



1 Introduction

Background and motivation. Due to the enormous potential of preprocessing to speed up the algorithmic solution to combinatorial problems [1, 2, 3, 43, 44], a large body of work in theoretical computer science is devoted to understanding its power and limitations. Using the notion of *kernelization* [4, 20, 21, 25, 29, 33] from parameterized complexity [12, 15] it is possible to formulate a guarantee on the size of the instance after preprocessing based on the parameter of the original instance. Under this model, a good preprocessing algorithm is a *kernelization algorithm*: given a parameterized instance (x, k) , it outputs an equivalent instance (x', k') of the same decision problem such that $|x'| + k' \leq f(k)$ for some function f that bounds the size of the kernel. Research into kernelization led to deep algorithmic insights, including connections to protrusions and finite-state properties [5], well-quasi ordering [22], and matroids [30]; these positive results were complemented by impossibility results [13, 16, 30] delineating the boundaries of tractability.

Results on kernelization led to profound insights into the limitations of polynomial-time data compression for NP-hard problems. However, as recently advocated [14], the definition of kernelization only gives guarantees on the *size* of the instance after preprocessing, which



© Benjamin Merlin Bumpus, Bart M. P. Jansen, and Jari J. H. de Kroon; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 30; pp. 30:1–30:15

Leibniz International Proceedings in Informatics



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

does not directly correspond to the running time of subsequently applied algorithms. If the preprocessed instance is not solved by brute force, but via a fixed-parameter tractable algorithm whose running time is of the form $f(k) \cdot n^{\mathcal{O}(1)}$, then the exponential dependence in the running time is on the value of the *parameter* k , which is not guaranteed to decrease via kernelization. In fact, if $P \neq NP$ then no polynomial-time preprocessing algorithm can guarantee to always decrease the parameter of an NP-hard fixed-parameter tractable problem, as iterating the preprocessing algorithm would lead to its solution in polynomial time. In this work, we develop a new analysis of preprocessing aimed at reducing the search space of the follow-up algorithm. We apply this framework to combinatorial optimization problems on graphs, whose goal is to find a minimum vertex-subset satisfying certain properties. The main idea behind the framework is to define formally what it means for a vertex to be *essential* for making reasonable solutions to the problem, and to prove that an efficient preprocessing algorithm can detect a subset of an optimal solution that contains all such essential vertices.

Before stating our results, we introduce and motivate the model. We consider vertex-subset minimization problems on (possibly directed) graphs, in which the goal is to find a minimum vertex subset having a certain property. Examples of the problems we study include VERTEX COVER, ODD CYCLE TRANSVERSAL, and DOMINATING SET. The analysis of such minimization problems, parameterized by the size of the solution, has played an important role in the literature (cf. [9, 17, 27, 38, 40]). Our starting point is the thesis that a good preprocessing algorithm should reduce the *search space*. Since many graph problems are known to be fixed-parameter tractable when parameterized by the size of the solution, we can reduce the search space of these FPT algorithms by finding one or more vertices which are part of an optimal solution, thereby decreasing the size of the solution still to be found in the reduced instance (i.e. the parameter value).

Since in general no polynomial-time algorithm can guarantee to identify at least one vertex that belongs to an optimal solution, the guarantee of the effectiveness of the preprocessing algorithm should be stated in a more subtle way. When solving problems by hand, one sometimes finds that certain vertices v are easily identified to belong to an optimal solution, as avoiding them would force the solution to contain a prohibitively large number of alternative vertices and therefore be suboptimal. Can an efficient preprocessing algorithm identify those vertices that cannot be avoided when making an optimal solution?

Since many NP-hard problems remain hard even when there is a unique solution [42], this turns out to be too much to ask as it would allow instances with unique solutions to be solved in polynomial time, which leads to $NP = RP$. We therefore have to relax the requirements on the preprocessing guarantee slightly, as follows. For an instance of a vertex-subset minimization problem Π on a graph G , we denote the minimum size of a solution on G by $\text{OPT}_{\Pi}(G)$. For a fixed $c \in \mathbb{R}_{\geq 1}$, we say a vertex $v \in V(G)$ is *c-essential* for Π on G if all feasible solutions $S \subseteq V(G)$ for Π whose total size is at most $c \cdot \text{OPT}_{\Pi}(G)$ contain v . Based on this notion, we can ask ourselves: can an efficient preprocessing algorithm identify part of an optimal solution if there is at least one *c-essential* vertex?

Phrased in this way, the algorithmic task becomes more tractable. For example, for the VERTEX COVER problem, selecting all vertices that receive the value 1 in an optimal half-integral solution to the LP-relaxation results in a set S which is contained in some optimal solution (by the Nemhauser-Trotter theorem [37], cf [12, §2.5]), and at the same time includes all 2-essential vertices: any vertex $v \notin S$ only has neighbors of value $\frac{1}{2}$ and 1, which implies that the set X of vertices other than v whose value in the LP relaxation is at least $\frac{1}{2}$, forms a feasible solution which avoids v . Its cardinality is at most twice the cost of the LP relaxation and therefore X is a 2-approximation. Hence S contains all 2-essential vertices.

This example shows that a preprocessing step that detects c -essential vertices without any additional information is sometimes possible. However, to be able to extend the scope of our results also to problems which *do not* have polynomial-time constant-factor approximations, we slightly relax the requirements on the preprocessing algorithm as follows. Let Π be a minimization problem on graphs whose solutions are vertex subsets and let $c \in \mathbb{R}_{\geq 1}$.

c -ESSENTIAL DETECTION FOR Π

Input: A graph G and integer k .

Task: Find a vertex set $S \subseteq V(G)$ such that:

G1 if $\text{OPT}_{\Pi}(G) \leq k$, then there is an optimal solution in G containing all of S , and

G2 if $\text{OPT}_{\Pi}(G) = k$, then S contains all c -essential vertices.

In this model, the preprocessing task is facilitated by supplying an additional integer k in the input. The correctness properties of the output S are formulated in terms of k . If $\text{OPT}_{\Pi}(G) \leq k$, then the set S is required to be part of an optimal solution. The upper bound on $\text{OPT}_{\Pi}(G)$ is useful to the algorithm: whenever the algorithm establishes that avoiding v would incur a cost of more than k , it is safe to add v to S . If $\text{OPT}_{\Pi}(G) = k$, then the algorithm should guarantee that S contains all c -essential vertices. Knowing a lower bound on $\text{OPT}_{\Pi}(G)$ is useful to the algorithm in case it can establish that any optimal solution containing v can be transformed into one avoiding v whose cost is $(c - 1) \cdot k$ larger, which yields a c -approximation if $(c - 1) \cdot k \leq (c - 1) \text{OPT}_{\Pi}(G)$. Hence vertices for which such a replacement exists are not c -essential and may safely be left out of S .

Results. We present polynomial-time algorithms for c -ESSENTIAL DETECTION FOR Π for a range of vertex-deletion problems Π and small values of c ; typically $c \in \{2, 3\}$. Example applications include VERTEX COVER and FEEDBACK VERTEX SET, and also CHORDAL VERTEX DELETION (for which no $O(1)$ -approximation is known), ODD CYCLE TRANSVERSAL (for which no $O(1)$ -approximation exists, assuming the Unique Games Conjecture [28, 45]), and even DIRECTED ODD CYCLE TRANSVERSAL (which is $W[1]$ -hard parameterized by solution size [35]).

The model of c -ESSENTIAL DETECTION FOR Π is chosen such that the detection algorithms whose correctness is formulated with respect to the value of k , can be used as a preprocessing step to optimally solve vertex-subset problems without any knowledge of the optimum. Let $\mathcal{E}_c^{\Pi}(G)$ denote the set of c -essential vertices in G , which is well-defined since all optimal solutions contain all c -essential vertices. By using a preprocessing step that detects a superset of the c -essential vertices in the solution, we can effectively improve the running-time guarantee for FPT algorithms parameterized by solution size from $f(\text{OPT}_{\Pi}(G)) \cdot |V(G)|^{\mathcal{O}(1)}$, to $f(\text{OPT}_{\Pi}(G) - |\mathcal{E}_c^{\Pi}(G)|) \cdot |V(G)|^{\mathcal{O}(1)}$. This leads to the following results.

► **Theorem 1.1.** *For each problem Π with coefficient c and parameter dependence f listed in Table 1 that is not $W[1]$ -hard, there is an algorithm that, given a graph G , outputs an optimal solution in time $f(\ell) \cdot |V(G)|^{\mathcal{O}(1)}$, where $\ell := \text{OPT}_{\Pi}(G) - |\mathcal{E}_c^{\Pi}(G)|$ is the number of vertices in an optimal solution which are not c -essential.*

Hence the running time for solving these problems does not depend on the total size of an optimal solution, only on the part of the solution that does not consist of c -essential vertices. The theorem effectively shows that by employing c -ESSENTIAL DETECTION FOR Π as a preprocessing step, the size of the search space no longer depends on the total solution size but only on its non-essential vertices.

■ **Table 1** For each problem Π , there is a polynomial-time algorithm for c -ESSENTIAL DETECTION for the stated value of c . Combined with the state of the art algorithm for the natural parameterization, this leads to an algorithm solving the problem in time $f(\ell) \cdot |V(G)|^{\mathcal{O}(1)}$ where $\ell = \text{OPT}_{\Pi}(G) - |\mathcal{E}_c^{\Pi}(G)|$.

Problem	c	$f(\ell)$	Reference
VERTEX COVER	2	1.2738^{ℓ}	[9]
FEEDBACK VERTEX SET	2	2.7^{ℓ}	[31]
DIRECTED FEEDBACK VERTEX SET	2	$4^{\ell} \cdot \ell!$	[10]
ODD CYCLE TRANSVERSAL	2	2.3146^{ℓ}	[34]
DIRECTED ODD CYCLE TRANSVERSAL	3	W[1]-hard	[35]
CHORDAL VERTEX DELETION	13	$2^{\mathcal{O}(\ell \log \ell)}$	[8]

We also prove limitations to this approach. Assuming $\text{FPT} \neq \text{W}[1]$, for DOMINATING SET, PERFECT DELETION (in which the goal is to obtain a perfect graph by vertex deletions) and WHEEL-FREE DELETION, there is no polynomial-time algorithm for c -ESSENTIAL DETECTION for any $c \in \mathbb{R}_{\geq 1}$. In fact, we can even rule out such algorithms running in time $f(k) \cdot |V(G)|^{\mathcal{O}(1)}$. These results are based on FPT-inapproximability results for DOMINATING SET [40] and existing reductions [26, 32] to the mentioned vertex-deletion problems.

Techniques. The main work lies in the algorithms for c -ESSENTIAL DETECTION, which are all based on covering/packing duality for forbidden induced subgraphs to certain graph classes, or variations thereof in terms of (integer) solutions to certain linear programs and their (integer) duals. To understand the relation between detecting essential vertices and covering/packing duality, consider the ODD CYCLE TRANSVERSAL problem (OCT). Following the argumentation for the classic Erdős-Pósa theorem [18], in general there is no constant c such that any graph either has an odd cycle transversal of size $c \cdot k$, or a packing of k vertex-disjoint odd cycles. However, we show that a linear packing/covering relation holds in the following slightly different setting. If $G - v$ is bipartite (so all odd cycles of G intersect v), then the minimum size of an OCT avoiding v equals the maximum cardinality of a packing of odd cycles which pairwise intersect only at v . We can leverage this statement to prove that any vertex v which is not at the center of a flower (see Definition 3.1) of more than $\text{OPT}_{\text{oct}}(G)$ odd cycles, is not 2-essential: for any optimal solution X containing v , the graph $G' := G - (X \setminus \{v\})$ becomes bipartite after removal of v and by assumption does not contain a packing of more than $\text{OPT}_{\text{oct}}(G)$ odd cycles pairwise intersecting at v . So by covering/packing duality on G' , it has an OCT X' of size at most $\text{OPT}_{\text{oct}}(G)$ avoiding v , so that $(X \setminus \{v\}) \cup X'$ is a 2-approximation in G which avoids v , showing that v is not 2-essential. Since v is clearly contained in an optimal solution whenever there is a flower of more than $\text{OPT}_{\text{oct}}(G)$ odd cycles centered at v , this yields a method for c -ESSENTIAL DETECTION when using a known reduction [23] to MAXIMUM MATCHING to test for a flower of odd cycles.

Organization. After presenting preliminaries in Section 2, we give algorithms to detect essential vertices based on covering/packing duality in Section 3 and based on integrality gaps in Section 4. In Section 5 we show how these detection subroutines can be used to improve the parameter dependence of FPT algorithms parameterized by solution size. The lower bounds are presented in Section 6. The investigation of c -essential vertices has close connections to the area of perturbation stability, which we briefly explore in Section 7. We conclude in Section 8. Due to space limitations, proofs of statements marked (★) are deferred to the full version [7].

2 Preliminaries

We consider finite simple graphs, some of which are directed. We use standard notation for graph algorithms; any terms not defined here can be found in the textbook by Cygan et al. [12]. We consider vertex-deletion problems on graphs. For a graph class \mathcal{C} , a \mathcal{C} -modulator in a graph G is a vertex set $S \subseteq V(G)$ such that $G - S \in \mathcal{C}$. The minimum size of a \mathcal{C} -modulator in G is denoted $\text{OPT}_{\mathcal{C}}(G)$. The corresponding minimization problem is defined as follows.

\mathcal{C} -DELETION

Input: A graph G .

Task: Find a minimum-size vertex-subset $S \subseteq V(G)$ such that $G - S \in \mathcal{C}$.

Throughout this paper we consider hereditary graph classes \mathcal{C} . These can be characterized by a (possibly infinite) set of forbidden induced subgraphs denoted $\text{forb}(\mathcal{C})$. The \mathcal{C} -DELETION problem is equivalent to finding a minimum set $S \subseteq V(G)$ such that no induced subgraph of $G - S$ is isomorphic to a graph in $\text{forb}(\mathcal{C})$. We say that such a set S hits all graphs from $\text{forb}(\mathcal{C})$ in G . For these classes the vertex set $V(G)$ is a trivial \mathcal{C} -modulator (since the empty graph is in all hereditary classes), which ensures that the task of finding a smallest modulator is always well-defined.

A graph is perfect if for every induced subgraph the size of a largest clique is equivalent to its chromatic number. Equivalently, a graph is perfect if it has no induced cycle of odd length at least five or its edge complement (cf. [24]). A graph is chordal if it has no induced cycle of length at least four. A graph is bipartite if its vertex set can be partitioned into two independent sets, or equivalently, it does not contain an odd-length cycle. Given a graph G and a set $T \subseteq V(G)$, a T -path is a path with at least one edge with both endpoints contained in T . A T -path is odd if it has an odd number of edges. For $u, v \in V(G)$, a (u, v) -separator is a set $S \subseteq V(G) \setminus \{u, v\}$ that disconnects u from v . If G is a directed graph, then in $G - S$ there is no directed path from u to v instead.

3 Positive results via Packing Covering

In this section we provide polynomial-time algorithms for c -ESSENTIAL DETECTION FOR II for various problems II. The case for the VERTEX COVER problem was given in Section 1. The results in this section are all based on packing/covering duality (cf. [11], [41, §73]). Towards this end, we generalize the notion of *flowers*, which played a key role in kernelization for FEEDBACK VERTEX SET [6]. While flowers were originally formulated as systems of cycles (forbidden structures for FEEDBACK VERTEX SET) pairwise intersecting in a single common vertex, we generalize the notion to near-packings of arbitrary structures here.

► **Definition 3.1.** *Let \mathcal{F} be a set of graphs. For a graph G and $v \in V(G)$, a (v, \mathcal{F}) -flower with p petals in G is a set $\{C_1, C_2, \dots, C_p\}$ of induced subgraphs of G such that each C_i (with $i \in [p]$) is isomorphic to some member of \mathcal{F} and such that $V(C_i) \cap V(C_j) = \{v\}$ for all distinct $i, j \in [p]$. The \mathcal{F} -flower number of a vertex $v \in V(G)$, denoted $\Gamma_{\mathcal{F}}(G, v)$, is the largest integer p for which there is a (v, \mathcal{F}) -flower in G with p petals.*

We show a general theorem for finding 2-essential vertices for \mathcal{C} -deletion if a maximum $(v, \text{forb}(\mathcal{C}))$ -flower can be computed in polynomial-time. It applies to those classes \mathcal{C} where graphs with $G - v \in \mathcal{C}$ obey a min-max relation between \mathcal{C} -modulators avoiding v and packings of forbidden induced subgraphs intersecting only at v .

► **Theorem 3.2.** *Let \mathcal{C} be a hereditary graph class such that, for any graph G and vertex $v \in V(G)$ with $G - v \in \mathcal{C}$, the minimum size of a \mathcal{C} -modulator avoiding v in G equals $\Gamma_{\text{forb}(\mathcal{C})}(G, v)$. Suppose there exists a polynomial-time algorithm \mathcal{A} that, given a graph G and vertex $v \in V(G)$, computes $\Gamma_{\text{forb}(\mathcal{C})}(G, v)$. Then there is a polynomial-time algorithm that solves 2-ESSENTIAL DETECTION FOR \mathcal{C} -DELETION.*

Proof. Apply algorithm \mathcal{A} to each vertex $v \in V(G)$ and let S be the set of vertices for which it finds that $\Gamma_{\text{forb}(\mathcal{C})}(G, v) > k$. We argue that Properties G1 and G2 are satisfied. If $\text{OPT}_{\mathcal{C}}(G) \leq k$, then every vertex in S is contained in every optimal solution for G since a size- k solution cannot hit a flower of $k + 1$ petals from $\text{forb}(\mathcal{C})$ without using v . Therefore Property G1 is satisfied. Next suppose that $\text{OPT}_{\mathcal{C}}(G) = k$ and let X be an optimal solution. We argue that each vertex $v \notin S$ is not 2-essential. Clearly this holds for any vertex $v \notin X$, so suppose that $v \in X$. Note that for every vertex $v \notin S$ we have $\Gamma_{\text{forb}(\mathcal{C})}(G, v) \leq k$, which implies that $\Gamma_{\text{forb}(\mathcal{C})}(G', v) \leq k$ where $G' := G - (X \setminus \{v\})$. Note that since $G' - v \in \mathcal{C}$, by assumption there exists a \mathcal{C} -modulator $X' \subseteq V(G') \setminus \{v\}$ in G' of size $\Gamma_{\text{forb}(\mathcal{C})}(G', v) \leq k$. Observe that $(X \setminus \{v\}) \cup X'$ is a \mathcal{C} -modulator in G of size at most $2k$ that avoids v and therefore v is not 2-essential. ◀

Theorem 3.2 allows us to conclude the following result for FEEDBACK VERTEX SET (FVS) and its directed variant (DFVS) using Gallai's theorem and Menger's theorem, respectively.

► **Lemma 3.3 (★).** *There are polynomial-time algorithms for 2-ESSENTIAL DETECTION FOR Π for $\Pi \in \{\text{FVS}, \text{DFVS}\}$.*

Next, we consider ODD CYCLE TRANSVERSAL (OCT), which corresponds to \mathcal{C} -DELETION where \mathcal{C} is the class of bipartite graphs and $\text{forb}(\mathcal{C})$ consists of all odd cycles. In order to apply Theorem 3.2 to OCT, we first argue that the class of bipartite graphs satisfies the preconditions. The proof is similar to that of Geelen et al. [23, Lemma 11] who reduce the problem of packing odd cycles containing v to a matching problem. We note that, although their result can be used to obtain a 3-essential detection algorithm, we will show (Lemma 3.7) how to efficiently detect 2-essential vertices as well. If the graph resulting from their construction has a large matching, then there is a large $(v, \text{forb}(\mathcal{C}))$ -flower. If on the other hand there is no large matching, then the Tutte-Berge formula is used to obtain a set of size $2k$ that hits all the odd cycles passing through v . We show that if the graph $G - v$ is bipartite instead, then this second argument can be improved to obtain a hitting set of size k by noting that the lack of a large matching implies that there is a small vertex cover due to König's theorem. This is below, using the viewpoint that odd cycles in G correspond to odd T -paths in $G - v$ for $T = N_G(v)$.

► **Lemma 3.4.** *For any undirected graph G and set $T \subseteq V(G)$, a maximum packing of odd T -paths can be computed in polynomial time. Moreover, if G is bipartite then the cardinality of a maximum packing of odd T -paths is equal to the minimum size of a vertex set which intersects all odd T -paths.*

Proof. We reduce to matching as in [23, Lemma 11]. Construct a graph H as follows. For each $v \in V(G) \setminus T$, let $v' \notin V(G)$ be a copy of v . Let $V(H) = V(G) \cup \{v' \mid v \in V(G) \setminus T\}$ and $E(H) = E(G) \cup \{u'v' \mid uv \in E(G - T)\} \cup \{vv' \mid v \in V(G) \setminus T\}$. Note that the graph H consists of the disjoint union of G and a copy of $G - T$, with an added edge between $v \in V(G) \setminus T$ and its copy v' . Geelen et al. [23] mention that there is a 1-1 correspondence between odd T -paths in G and certain augmenting paths in H . For completeness we give a self-contained argument.

▷ **Claim 3.5.** Graph G contains k vertex-disjoint odd T -paths if and only if H has a matching M of size $|V(G) \setminus T| + k$. Furthermore, given a matching M in H of size $|V(G) \setminus T| + k$ we can compute a set of k vertex-disjoint odd T -paths in polynomial time.

Proof. (\Rightarrow) Let $\mathcal{P} = (P_1, \dots, P_k)$ be a set of k vertex-disjoint odd T -paths in G . Consider a path $P = (v_1, \dots, v_{2\ell}) \in \mathcal{P}$, where $\ell \geq 1$. First note that we can assume that $V(P) \cap T = \{v_1, v_{2\ell}\}$, since if $v_i \in T$ for some $1 < i < 2\ell$, then either (v_1, \dots, v_i) or $(v_i, \dots, v_{2\ell})$ is an odd T -path and we can update \mathcal{P} accordingly. Construct a matching M in H as follows. For any path $P = (v_1, \dots, v_{2\ell}) \in \mathcal{P}$, add the edges $v_1v_2, v'_2v'_3, \dots, v_{2\ell-1}v_{2\ell}$, alternating between original vertices and copy vertices. This is possible as P is of odd length. For any vertex in $u \in V(G) \setminus T$ that is not contained in an odd T -path, we add uu' to M . Observe that at least $2|V(G) \setminus T| + 2k$ vertices are matched, therefore $|M| \geq |V(G) \setminus T| + k$ as desired.

(\Leftarrow) Let M be a matching of size $|V(G) \setminus T| + k$. If M contains both uv and $u'v'$ for $u, v \in V(G) \setminus T$, then update M by removing them and inserting uu' and vv' instead. If for $v \in V(G) \setminus T$ only one of v and v' is matched, and it is not matched to its copy, then match it to its copy instead. Afterwards let $E' := \{uv \in E(G) \mid uv \in M \vee u'v' \in M\}$. Observe that in $G[E']$, each vertex in $V(G) \setminus T$ has degree 0 or 2. For each $v \in V(G) \setminus T$ such that v has degree 0 in $G[E']$, add vv' to M if it is not in already. Note that all vertices of $H - T$ are matched. It follows that at least $2k$ vertices in T are matched by M and they have degree 1 in $G[E']$. Observe that $G[E']$ is a collection of paths and cycles with all degree-1 vertices in T . We get that there are k T -paths in G that are of odd length by construction (every even numbered edge in $G[E']$ originated from the copy part of H). Note that we can find them in polynomial time. \triangleleft

Since a maximum matching can be computed in polynomial time, by the claim above we get that a maximum packing of vertex-disjoint odd T -paths can be computed in polynomial time. Next we prove the second part of the statement.

▷ **Claim 3.6.** If G is bipartite and a maximum matching M in H has size $|V(G) \setminus T| + k$, then there is a set $S \subseteq V(G)$ of size at most k such that $G - S$ has no odd-length T -path.

Proof. Observe that since G is bipartite, the graph H is bipartite as well. By König's theorem [41, Theorem 16.2], H has a vertex cover X of size $|V(G) \setminus T| + k$. Let $S = (X \cap T) \cup \{u \mid \{u, u'\} \subseteq X\}$. Note that for each $u \in V(G) \setminus T$, at least one of $u \in X$ or $u' \in X$ must hold to cover the edge uu' , thereby already accounting for $|V(G) \setminus T|$ vertices of the cover. It follows that $|S| \leq k$. We argue that S hits all odd-length T -paths in G .

For the sake of contradiction suppose there is some odd-length T -path from $t_1 \in T$ to $t_2 \in T \setminus \{t_1\}$ in $G - S$. Let $P = (t_1, v_1, \dots, v_{2\ell}, t_2)$ be a (not necessarily induced) shortest odd T -path. Note that neither of t_1 and t_2 belong to X by construction of S . Furthermore $P \setminus \{t_1, t_2\} \subseteq V(G) \setminus T$, as otherwise we could construct a shorter odd-length T -path. Consider the vertices $V(P) \cup \{v'_1, \dots, v'_{2\ell}\} \subseteq V(H)$. By construction of S , the vertex cover X has exactly one of v_i and v'_i for each $i \in [2\ell]$. Observe that for it to be a vertex cover, X must contain vertices of $P \setminus \{t_1, t_2\}$ and their copies in an alternating fashion since for each edge v_i, v_{i+1} of P , the graph H contains edges $v_iv'_i, v_{i+1}v'_{i+1}, v_iv_{i+1}, v'_iv'_{i+1}$. Without loss of generality, let $v_1 \in X$. It follows that $v_{2\ell} \notin X$. But this contradicts that X is a vertex cover as $v_{2\ell}t_2$ is not covered. We conclude that S hits all odd-length T -paths in G . \triangleleft

By Claim 3.5, a maximum packing of k of vertex-disjoint odd T -paths in G implies a matching in H of size $|V(G) \setminus T| + k$. By Claim 3.6, we can create a set of size at most k that intersects all odd T -paths in the bipartite graph G . Clearly such a set has size at least k . It follows that if G is bipartite, then the cardinality of a maximum packing of odd

T -paths is equal to the minimum size of a vertex set which intersects all odd T -paths. (For completeness, we remark that this last property can also be derived from Menger's theorem: in a bipartite graph with bipartition into $A \cup B$, a T -path is odd if and only if its endpoints belong to different partite sets, so a maximum packing of odd T -paths is equivalent to a maximum set of vertex-disjoint paths between $A \cap T$ and $B \cap T$.) ◀

By observing that odd T -paths in $G - v$ directly correspond to flowers with odd cycles pairwise intersecting at v in G , Lemma 3.4 and Theorem 3.2 imply the following.

► **Lemma 3.7.** *There is a polynomial-time algorithm for 2-ESSENTIAL DETECTION FOR OCT.*

The DOCT problem corresponds to \mathcal{C} -deletion where $\text{forb}(\mathcal{C})$ consists of all directed cycles of odd length. Using Menger's theorem on an auxiliary graph, we can detect 3-essential vertices for this problem.

► **Lemma 3.8 (★).** *There is a polynomial-time algorithm for 3-ESSENTIAL DETECTION FOR DOCT.*

We cannot use the approach based on computing a maximum (v, \mathcal{F}) -flower for the CHORDAL DELETION problem; a simple reduction¹ from DISJOINT PATHS [39] shows that it is NP-hard to compute a maximum (v, \mathcal{F}) -flower when \mathcal{F} is the set of chordless cycles of length at least four. In the next section, we will therefore use an approach based on the linear-programming relaxation to deal with CHORDAL DELETION.

4 Positive results via Linear Programming

Consider the following natural linear program for \mathcal{C} -DELETION for hereditary graph classes \mathcal{C} . The LP corresponding to an input graph G is defined on the variables $(x_u)_{u \in V(G)}$, as follows.

\mathcal{C} -DELETION LP

Objective: minimize $\sum_{u \in V(G)} x_u$.

Subject to:

- $\sum_{u \in V(H)} x_u \geq 1$ for each induced subgraph H of G isomorphic to a graph in $\text{forb}(\mathcal{C})$,
- $0 \leq x_u \leq 1$ for each $u \in V(G)$.

In the corresponding integer program, the constraint $0 \leq x_u \leq 1$ is replaced by $x_u \in \{0, 1\}$. We say that a minimization LP has *integrality gap* at most c for some $c \in \mathbb{R}$ if the cost of an integer optimum is at most c times the cost of a fractional optimum. In general, the number of constraints in the \mathcal{C} -DELETION LP can be exponential in the size of the graph. Using the ellipsoid method (cf. [41]), this can be handled using a separation oracle: a polynomial-time algorithm that, given an assignment to the variables, outputs a violated constraint if one exists. It is well-known (cf. [41, Thm. 5.10]) that linear programs with an exponential number of constraints can be solved in polynomial time using a polynomial-time separation oracle. To detect essential vertices, the integrality gap of a slightly extended LP will be crucial. We define the v -AVOIDING \mathcal{C} -DELETION LP for a graph G and distinguished vertex $v \in V(G)$ as the \mathcal{C} -DELETION LP with the additional constraint that $x_v = 0$. Hence its integral solutions correspond to \mathcal{C} -modulators avoiding v .

¹ Starting from an instance $(G, (s_1, t_1), \dots, (s_\ell, t_\ell))$ of DISJOINT PATHS satisfying $s_i t_i \notin E(G)$ for all $i \in [\ell]$ (which is without loss of generality), insert a vertex v adjacent to $A = \bigcup_{i=1}^{\ell} \{s_i, t_i\}$ and insert all edges between vertices in A except $s_i t_i$ for each $i \in [\ell]$.

► **Theorem 4.1 (★).** *Let \mathcal{C} be a hereditary graph class such that for each graph G and $v \in V(G)$ satisfying $G - v \in \mathcal{C}$, the integrality gap of v -AVOIDING \mathcal{C} -DELETION on G is at most $c \in \mathbb{R}_{\geq 1}$. If there is a polynomial-time separation oracle for the \mathcal{C} -DELETION LP, then there is a polynomial-time algorithm for $(c + 1)$ -ESSENTIAL DETECTION FOR \mathcal{C} -DELETION.*

Using known results on covering versus packing for chordless cycles in near-chordal graphs, the approach above can be used to detect essential vertices for CHORDAL DELETION. For the class of chordal graphs, the corresponding set of forbidden induced subgraphs is the class hole of all holes, i.e., induced chordless cycles of length at least four.

► **Lemma 4.2 (★).** *There is a polynomial-time algorithm for 13-ESSENTIAL DETECTION FOR CHORDAL DELETION.*

5 Consequences for Parameterized Algorithms

In this section we show how the algorithms for c -ESSENTIAL DETECTION from the previous section can be used to solve \mathcal{C} -DELETION for various classes \mathcal{C} , despite the fact that the detection algorithms only work when certain guarantees on k are met. The main theorem connecting the detection problem to solving \mathcal{C} -DELETION is the following.

► **Theorem 5.1.** *Let \mathcal{A} be an algorithm that, given a graph G and an integer k , runs in time $f(k) \cdot |V(G)|^{\mathcal{O}(1)}$ for some non-decreasing function f and returns a minimum-size \mathcal{C} -modulator if there is one of size at most k . Let \mathcal{B} be a polynomial-time algorithm for c -ESSENTIAL DETECTION FOR \mathcal{C} -DELETION. Then there is an algorithm that, given a graph G , outputs a minimum-size \mathcal{C} -modulator in time $f(\ell) \cdot |V(G)|^{\mathcal{O}(1)}$, where $\ell = \text{OPT}_{\mathcal{C}}(G) - |\mathcal{E}_c(G)|$ is the c -non-essentiality.*

Proof. First we describe the algorithm as follows. For each $0 \leq k \leq |V(G)|$, let S_k be the result of running \mathcal{B} on (G, k) , let $G_k := G - S_k$, and let $b_k := k - |S_k|$.

Letting L be the list of all triples (G_k, S_k, b_k) sorted in increasing order by their third component b_k , proceed as follows. For each $(G_k, S_k, b_k) \in L$, run \mathcal{A} on (G_k, b_k) to find a minimum \mathcal{C} -modulator $S_{\mathcal{A}}$ of size at most b_k , if one exists. If $|S_{\mathcal{A}}| = b_k$, then output $S_{\mathcal{A}} \cup S_k$ as a minimum \mathcal{C} -modulator in G .

To analyze the algorithm, we first argue it always outputs a solution. For the call with $k^* = \text{OPT}_{\mathcal{C}}(G)$, both conditions of the detection problem are met. Hence by Property G1 the set S_{k^*} is contained in a minimum modulator in G , so that $\text{OPT}_{\mathcal{C}}(G - S_{k^*}) = \text{OPT}_{\mathcal{C}}(G) - |S_{k^*}| = k^* - |S_{k^*}|$. Therefore graph $G_{k^*} = G - S_{k^*}$ has a modulator of size at most $b_{k^*} = \text{OPT}_{\mathcal{C}}(G - S_{k^*})$ and none which are smaller, so that \mathcal{A} correctly outputs a modulator of size b_{k^*} . In turn, this causes the overall algorithm to terminate with a solution.

Having established that the algorithm outputs a solution, we proceed to show that it outputs a minimum-size modulator whenever it outputs a solution (which may be in an earlier iteration than for $k^* = \text{OPT}_{\mathcal{C}}(G)$). Let k' be the value of k that is reached when the algorithm outputs a solution $S_{\mathcal{A}} \cup S_{k'}$. Then we know:

1. algorithm \mathcal{A} found a minimum-size modulator $S_{\mathcal{A}}$ in $G_{k'}$ of size at most $b_{k'}$, and
2. the set $S_{\mathcal{A}} \cup S_{k'}$ is a modulator in G , since $S_{\mathcal{A}}$ is a modulator in $G_{k'} = G - S_{k'}$, and therefore $\text{OPT}_{\mathcal{C}}(G) \leq b_{k'} + |S_{k'}| = k'$.

To see that the algorithm is correct, notice that, since $\text{OPT}_{\mathcal{C}}(G) \leq k'$, the set $S_{k'}$ is contained in some minimum-size modulator for G (since \mathcal{B} satisfies Property G1). Hence $\text{OPT}_{\mathcal{C}}(G_{k'}) = \text{OPT}_{\mathcal{C}}(G) - |S_{k'}|$. Since \mathcal{A} outputs a minimum-size modulator if there is one of size at most b_k , we have $|S_{\mathcal{A}}| = \text{OPT}_{\mathcal{C}}(G) - |S_{k'}|$, so that $\mathcal{A}(G_{k'}) \cup S_{k'}$ is a feasible modulator of size $\text{OPT}_{\mathcal{C}}(G)$ and therefore optimal.

30:10 Search-Space Reduction via Essential Vertices

Now we prove the desired running-time bound. First of all, notice that we can determine the list L in polynomial time by running \mathcal{B} once for each value of k (which is at most $|V(G)|$). By how we sorted L , we compute $\mathcal{A}(G_k, b_k)$ only when $b_k \leq b_{k^*}$, as we argued above that if the algorithm has not already terminated, it does so after reaching $k^* = \text{OPT}_c(G)$. Hence the calls to algorithm \mathcal{A} are for values of the budget b_k which satisfy $b_k \leq b_{k^*}$. We bound the latter, as follows.

Since $k^* = \text{OPT}_c(G)$, the set S_{k^*} found by \mathcal{B} is a superset of the set $\mathcal{E}_c(G)$ of all of the c -essential vertices in G (Property G2). This means that we have

$$b_{k^*} = \text{OPT}_c(G) - |S_{k^*}| \leq \text{OPT}_c(G) - |\mathcal{E}_c(G)| = \ell,$$

so the parameter of each call to \mathcal{A} is at most ℓ , giving the total time bound $f(\ell) \cdot |V(G)|^{\mathcal{O}(1)}$. ◀

Theorem 1.1 now follows from Theorem 5.1 via the algorithms for c -ESSENTIAL DETECTION given in the previous sections and the state-of-the-art algorithms for the natural parameterizations listed in Table 1. Although the latter may be originally stated for the decision version, using self-reduction they can easily be adapted to output a minimum solution if there is one of size at most k .

6 Hardness results

Given the positive results we saw in Sections 3 and 4, it is natural to seek problems Π for which c -ESSENTIAL DETECTION FOR Π is intractable. Here we show that c -ESSENTIAL DETECTION FOR DOMINATING SET is intractable for any $c \in \mathcal{O}(1)$ and then use this as a starting point to prove similar results for HITTING SET, PERFECT DELETION, and WHEEL-FREE DELETION.

A dominating set is a vertex set whose closed neighborhood is the entire graph. The domination number of a graph is the size of a minimum dominating set. The starting point for our reductions is the following result which states that it is $W[1]$ -hard to solve DOMINATING SET parameterized by solution size even on instances which have “solution-size gaps”.

► **Lemma 6.1** ([40], cf. [19, Thm. 4]). *Let $F, f: \mathbb{N} \rightarrow \mathbb{N}$ be any computable functions. Assuming $\text{FPT} \neq W[1]$, there does not exist an algorithm that, given a graph G and integer k , runs in time $f(k) \cdot |V(G)|^{\mathcal{O}(1)}$ and distinguishes between the following two cases:*

- *Completeness: G has a dominating set of size k .*
- *Soundness: Every dominating set of G is of size at least $k \cdot F(k)$.*

All of our reductions in this section share a leitmotif. We start with a *gap* instance (G, k) of DOMINATING SET and map it to an instance G' of c -ESSENTIAL DETECTION FOR Π (for appropriate Π) equipped with a distinguished vertex v^* with the following property: (1) if G has domination number at most k , then no optimal solution in G' contains v^* ; (2) if G has domination number strictly greater than $c \cdot F(k)$ (for some appropriate F), then v^* is contained in every solution of size at most $c \cdot F(k)$ in G' . Thus our hardness results will follow by combining reductions of this kind with Lemma 6.1.

► **Lemma 6.2** (★). *There is a polynomial-time algorithm R that, given a graph G and integer k , outputs a graph $R(G, k)$ containing a distinguished vertex v^* such that:*

- *if G has dominating number at most k , then the domination number of $R(G, k)$ is exactly k and every optimal dominating set avoids v^* ;*
- *if G has domination number strictly greater than $c \cdot (k + 1)$ for some $c \in \mathbb{R}_{\geq 1}$, then $R(G, k)$ has domination number $k + 1$ and the distinguished vertex v^* is contained in all $R(G, k)$ -dominating sets of size at most $c \cdot (k + 1)$.*

Proof sketch. The graph $R(G, k)$ is defined formally as follows:

1. initialize $R(G, k)$ as the graph on vertex set $\{v_i \mid v \in V(G), 0 \leq i \leq k\}$ with edges $\{v_i u_j \mid uv \in E(G), 0 \leq i, j \leq k\}$,
2. for each $i \in [k]$ insert an apex a_i which is adjacent to $\{v_i \mid v \in V(G)\}$,
3. insert a vertex v^* which is adjacent to $\{v_i \mid v \in V(G), 0 \leq i \leq k\}$.

The proof that $R(G, k)$ has the desired properties is given in the full version [7], together with a figure of the construction. ◀

Lemma 6.1 combined with the reduction provided by Lemma 6.2 yields the following.

► **Theorem 6.3.** *Unless $FPT = W[1]$, there is no FPT-time algorithm for c -ESSENTIAL DETECTION FOR DOMINATING SET parameterized by k for any $c \in \mathbb{R}_{\geq 1}$.*

Proof. Suppose such an algorithm \mathcal{A} exists for c ; we will use it with Lemma 6.1 to show $FPT = W[1]$ for the function $F(k) = c(k + 1)$.

Given an input (G, k) in which the goal is to distinguish between the completeness and soundness cases, the algorithm proceeds as follows. Using the reduction R of Lemma 6.2, consider the graph $R(G, k)$ and let S be the output of an algorithm for c -ESSENTIAL DETECTION FOR DOMINATING SET on the pair $(R(G, k), k + 1)$; note that the solution size for which we ask is $k + 1$ rather than k . Without loss of generality we may assume $k \geq 2$, as the distinction can trivially be made otherwise. We will show that in the completeness case we have $v^* \notin S$, while in the soundness case we have $v^* \in S$, which allows us to distinguish between these cases by checking whether v^* belongs to the output of $\mathcal{A}(R(G, k), k + 1)$.

For the completeness case, suppose G has domination number at most k . Then, by Lemma 6.2, so does $R(G, k)$. This means that Property G1 holds for the call to $\mathcal{A}(R(G, k), k + 1)$, so that there is some optimal solution S' of size k which contains S and hence we have $v^* \notin S$ by Lemma 6.2.

For the soundness case, suppose G has domination number at least $k \cdot F(k) = c(k + 1)k > c(k + 1)$ (we use $k \geq 2$ here). Then by Lemma 6.2, graph $R(G, k)$ has domination number $k + 1$ and v^* is contained in all its dominating sets of size at most $c(k + 1)$. In other words: v^* is c -essential in $R(G, k)$. Consequently, $v^* \in S$ by Property G2 since the argument $k + 1$ we supplied to \mathcal{A} coincides with the optimum in $R(G, k)$ in this case.

If \mathcal{A} runs in FPT-time, then the overall procedure runs in FPT-time which implies $FPT = W[1]$ by Lemma 6.1. ◀

Combining Lemma 6.2 with the standard reduction S from DOMINATING SET to HITTING SET (where hyperedges are given by closed neighborhoods) yields a composite mapping $S \circ R$ which relates c -essentiality to gaps in solution quality in much the same way as R did. We leverage this together with known reductions from HITTING SET to PERFECT DELETION [26] and WHEEL-FREE DELETION [32] to show the following.

► **Theorem 6.4 (★).** *Unless $FPT = W[1]$, both c -ESSENTIAL DETECTION FOR PERFDEL and c -ESSENTIAL DETECTION FOR WHEELDEL do not admit FPT-time algorithms parameterized by k for any $c \in \mathbb{R}_{\geq 1}$.*

7 Connections to Perturbation Resilience

In the area of perturbation resilience [36] there is a notion of so-called c -perturbation resilient instances to optimization problems. Roughly, these are instances G in which there is a unique optimal solution S which far outperforms (by a factor of c) every other solution S' in G .

More formally, for a vertex-weighted graph minimization problem Π whose solution is a vertex-subset, we say that an instance $(G, w: V(G) \rightarrow \mathbb{N})$ with a *unique* optimal solution S is *c-perturbation resilient* if for any perturbed weight function w' satisfying $w(v) \leq w'(v) \leq c \cdot w(v)$ for all $v \in V(G)$, the instance $(G, w'(x))$ has a unique optimal solution and furthermore this solution is S . (Of course, one can define an analogous notion for maximization problems as well and what follows in this section applies to both.)

Classes of c -perturbation resilient instances have been shown to be “islands of tractability” where many intractable problems become polynomial-time-solvable [36] and the suggestive intuition behind this fact is that perturbation resilient instances have unique optima which “stand out” and are “obvious” in some sense. Viewing stability through the lens of parameterized complexity, it is natural to ask whether one can quantify in an algorithmically useful way how distant an input is from being stable. The following proposition supports our claim that the *non-essentiality* (recall Theorem 5.1) is a good such measure since on $(> c)$ -stable inputs, the c -non-essentiality is smallest possible (namely it is 0).

► **Proposition 7.1.** *Given constants $c' > c \geq 1$, if G is a c' -stable input to a graph optimization problem Π whose solutions are vertex-subsets, then $\mathcal{E}_c^\Pi(G)$ is the unique optimum of G .*

Proof. Consider the unique optimum S for Π on G . If S' is a c -approximation for Π on G , then we must have $S' = S$ (since otherwise G would not be c' -stable). As a consequence we know that every vertex of S must be c -essential and hence we have $S \subseteq \mathcal{E}_c(G) \subseteq S$. ◀

Proposition 7.1 and Theorem 5.1 allow us to immediately deduce that any algorithm for c -ESSENTIAL DETECTION solves all $(> c)$ -perturbation resilient instances exactly.

► **Corollary 7.2.** *Given a minimization problem Π , any algorithm for c -ESSENTIAL DETECTION FOR Π solves $(> c)$ -stable instances exactly.*

8 Conclusion

We introduced the notion of c -essential vertices for vertex-subset minimization problems on graphs, to formalize the idea that a vertex belongs to all *reasonable* solutions. Using a variety of approaches centered around the theme of covering/packing duality, we gave polynomial-time algorithms that detect a subset of an optimal solution containing all c -essential vertices, which decreased the search space of parameterized algorithms from exponential in the size of the solution, to exponential in the number of non-essential vertices in the solution.

Throughout the paper we have restricted ourselves to working with unweighted problems. However, many of the same ideas can be applied in the setting where each vertex has a positive integer weight of magnitude $\mathcal{O}(|V(G)|^{\mathcal{O}(1)})$ and we search for a minimum-weight solution. Since integral vertex weights can be simulated for many problems by making twin-copies of a vertex, our approach can be extended to WEIGHTED VERTEX COVER, WEIGHTED ODD CYCLE TRANSVERSAL, and WEIGHTED CHORDAL DELETION.

Our results shed a new light on which instances of NP-hard problems can be solved efficiently. FPT algorithms for parameterizations by solution size show that instances are easy when their optimal solutions are small. Theorem 1.1 refines this view: it shows that instances with large optimal solutions can still be easy, as long as only a small number of vertices in the optimum is not c -essential.

We remark that there is an alternative route to algorithms for c -ESSENTIAL DETECTION, which is applicable to \mathcal{C} -DELETION problems which admit a constant-factor approximation. If there is a polynomial-time algorithm that, given a graph G and vertex v , outputs a

c -approximation for the problem of finding a minimum-size \mathcal{C} -modulator avoiding v , it can be used for c -ESSENTIAL DETECTION. A valid output S for the detection problem with input (G, k) is obtained by letting S contain all vertices for which the approximation algorithm outputs a v -avoiding modulator of size more than $c \cdot k$. Using this approach (cf. [22]) one can solve $\max_{F \in \mathcal{F}} |V(F)|^{\mathcal{O}(1)}$ -ESSENTIAL DETECTION FOR \mathcal{F} -MINOR-FREE DELETION for any finite family \mathcal{F} containing a planar graph. As the results for problems for which no constant-factor approximation exists are more interesting, we focused on those.

Our work opens up several questions for future work. Is the integrality gap for v -AVOIDING PLANAR VERTEX DELETION constant, when $G - v$ is planar? Can $\mathcal{O}(1)$ -ESSENTIAL DETECTION FOR PLANAR VERTEX DELETION be solved in polynomial time? Can 2-ESSENTIAL DETECTION FOR CHORDAL DELETION be solved in polynomial time? Can the constant c for which we can detect c -essential vertices be lowered below 2?

Considering a broader horizon, it would be interesting to investigate whether there are less restrictive notions than c -essentiality which can be used as the basis for guaranteed search-space reduction.




References

- 1 Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. Technical Report 16-44, ZIB, Takustr.7, 14195 Berlin, 2016. URL: <http://nbn-resolving.de/urn:nbn:de:0297-zib-60370>.
- 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 Jochen Alber, Nadja Betzler, and Rolf Niedermeier. Experiments on data reduction for optimal domination in networks. *Annals OR*, 146(1):105–117, 2006. doi:10.1007/s10479-006-0045-4.
- 4 Hans L. Bodlaender. Lower bounds for kernelization. In Marek Cygan and Pinar Heggernes, editors, *Parameterized and Exact Computation - 9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10-12, 2014. Revised Selected Papers*, volume 8894 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2014. doi:10.1007/978-3-319-13524-3_1.
- 5 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 629–638. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.46.
- 6 Hans L. Bodlaender and Thomas C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory Comput. Syst.*, 46(3):566–597, 2010. doi:10.1007/s00224-009-9234-2.
- 7 Benjamin Merlin Bumpus, Bart M. P. Jansen, and Jari J. H. de Kroon. Search-space reduction via essential vertices. *CoRR*, abs/2207.00386, 2022. arXiv:2207.00386.
- 8 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016. doi:10.1007/s00453-015-0014-x.
- 9 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.
- 10 Jianer Chen, Yang Liu, Songjian Lu, Barry O’Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008. doi:10.1145/1411509.1411511.
- 11 Maria Chudnovsky, Jim Geelen, Bert Gerards, Luis A. Goddyn, Michael Lohman, and Paul D. Seymour. Packing non-zero A -paths in group-labelled graphs. *Comb.*, 26(5):521–532, 2006. doi:10.1007/s00493-006-0030-1.
- 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 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.
- 14 Huib Donkers and Bart M. P. Jansen. Preprocessing to reduce the search space: Antler structures for feedback vertex set. In Lukasz Kowalik, Michal Pilipczuk, and Pawel Rzazewski, editors, *Graph-Theoretic Concepts in Computer Science - 47th International Workshop, WG 2021, Warsaw, Poland, June 23-25, 2021, Revised Selected Papers*, volume 12911 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2021. doi:10.1007/978-3-030-86838-3_1.
- 15 Rodney G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer Publishing Company, Incorporated, 2012.
- 16 Andrew Drucker. New limits to classical and quantum instance compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015. doi:10.1137/130927115.
- 17 Eduard Eiben, Robert Ganian, Thekla Hamm, and O-jeong Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. *J. Comput. Syst. Sci.*, 121:57–75, 2021. doi:10.1016/j.jcss.2021.04.005.
- 18 P. Erdős and L. Pósa. On independent circuits contained in a graph. *Canadian Journal of Mathematics*, 17:347–352, 1965. doi:10.4153/CJM-1965-035-8.
- 19 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/a13060146.
- 20 Michael R. Fellows. The lost continent of polynomial time: Preprocessing and kernelization. In *Proc. 2nd IWPEC*, pages 276–277, 2006. doi:10.1007/11847250_25.
- 21 Fedor Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
- 22 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar 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. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.
- 23 Jim Geelen, Bert Gerards, Bruce A. Reed, Paul D. Seymour, and Adrian Vetta. On the odd-minor variant of Hadwiger’s conjecture. *J. Comb. Theory, Ser. B*, 99(1):20–29, 2009. doi:10.1016/j.jctb.2008.03.006.
- 24 M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. ISSN. Elsevier Science, 2004.
- 25 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007. doi:10.1145/1233481.1233493.
- 26 Pinar Heggeres, Pim van ’t Hof, Bart M. P. Jansen, Stefan Kratsch, and Yngve Villanger. Parameterized complexity of vertex deletion into perfect graph classes. *Theor. Comput. Sci.*, 511:172–180, 2013. doi:10.1016/j.tcs.2012.03.013.
- 27 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC ’21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1757–1769. ACM, 2021. doi:10.1145/3406325.3451068.
- 28 Subhash Khot. On the power of unique 2-prover 1-round games. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.
- 29 Stefan Kratsch. Recent developments in kernelization: A survey. *Bull. EATCS*, 113, 2014. URL: <http://eatcs.org/beatcs/index.php/beatcs/article/view/285>.
- 30 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.
- 31 Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size k in $O^*(2.7^k)$ 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 971–989. SIAM, 2020. doi:10.1137/1.9781611975994.58.

- 32 Daniel Lokshantov. Wheel-free deletion is $W[2]$ -hard. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, volume 5018 of *Lecture Notes in Computer Science*, pages 141–147. Springer, 2008. doi:10.1007/978-3-540-79723-4_14.
- 33 Daniel Lokshantov, Neeldhara Misra, and Saket Saurabh. Kernelization - Preprocessing with a guarantee. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 129–161. Springer, 2012. doi:10.1007/978-3-642-30891-8_10.
- 34 Daniel Lokshantov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.
- 35 Daniel Lokshantov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. 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 2181–2200. SIAM, 2020. doi:10.1137/1.9781611975994.134.
- 36 Konstantin Makarychev and Yury Makarychev. Perturbation resilience. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 95–119. Cambridge University Press, 2020. doi:10.1017/9781108637435.008.
- 37 George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Math. Program.*, 8(1):232–248, 1975. doi:10.1007/BF01580444.
- 38 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.
- 39 Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.
- 40 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. *J. ACM*, 66(5):33:1–33:38, 2019. doi:10.1145/3325116.
- 41 A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.
- 42 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986. doi:10.1016/0304-3975(86)90135-0.
- 43 Karsten Weihe. Covering trains by stations or the power of data reduction. In *Algorithms and Experiments (ALEX98)*, pages 1–8, 1998. URL: <https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.2173>.
- 44 Karsten Weihe. On the differences between “Practical” and “Applied”. In Stefan Näher and Dorothea Wagner, editors, *Algorithm Engineering, 4th International Workshop, WAE 2000, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, volume 1982 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2000. doi:10.1007/3-540-44691-5_1.
- 45 Sebastian Wernicke. *On the algorithmic tractability of single nucleotide polymorphism (SNP) analysis and related problems*. diplom.de, 2014.

Width Helps and Hinders Splitting Flows

Manuel Cáceres   




Department of Computer Science,
University of Helsinki, Finland

Andreas Grigorjew   


Department of Computer Science,
University of Helsinki, Finland

Brendan Mumej  

School of Computing, Montana State University,
Bozeman, MT, USA

Alexandru I. Tomescu   

Department of Computer Science,
University of Helsinki, Finland

Massimo Cairo 

Department of Computer Science,
University of Helsinki, Finland

Shahbaz Khan   

Department of Computer Science and Engineering,
Indian Institute of Technology Roorkee, India

Romeo Rizzi  

Department of Computer Science,
University of Verona, Italy

Lucia Williams   

School of Computing, Montana State University,
Bozeman, MT, USA

Abstract

Minimum flow decomposition (MFD) is the NP-hard problem of finding a smallest decomposition of a network flow X on directed graph G into weighted source-to-sink paths whose superposition equals X . We focus on a common formulation of the problem where the path weights must be non-negative integers and also on a new variant where these weights can be negative. We show that, for acyclic graphs, considering the *width* of the graph (the minimum number of s - t paths needed to cover all of its edges) yields advances in our understanding of its approximability. For the non-negative version, we show that a popular heuristic is a $O(\log |X|)$ -approximation ($|X|$ being the total flow of X) on graphs satisfying two properties related to the width (satisfied by e.g., series-parallel graphs), and strengthen its worst-case approximation ratio from $\Omega(\sqrt{m})$ to $\Omega(m/\log m)$ for sparse graphs, where m is the number of edges in the graph. For the negative version, we give a $(\lceil \log \|X\| \rceil + 1)$ -approximation ($\|X\|$ being the maximum absolute value of X on any edge) using a power-of-two approach, combined with parity fixing arguments and a decomposition of unitary flows ($\|X\| \leq 1$) into at most width paths. We also disprove a conjecture about the linear independence of minimum (non-negative) flow decompositions posed by Kloster et al. [ALENEX 2018], but show that its useful implication (polynomial-time assignments of weights to a given set of paths to decompose a flow) holds for the negative version.

2012 ACM Subject Classification Theory of computation \rightarrow Network flows

Keywords and phrases Flow decomposition, approximation algorithms, graph width

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.31

Related Version *Full Version*: <https://arxiv.org/abs/2207.02136> [5]

Funding This work was partially funded by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFE BIO), partially by the Academy of Finland (grants No. 322595, 328877), and partially by the National Science Foundation (NSF) (grants No. 1661530,1759522).

1 Introduction

Minimum flow decomposition (MFD) is the problem of finding a smallest sized decomposition of a network flow X on directed graph $G = (V, E)$ into weighted source-to-sink paths whose superposition equals X . We focus on the case where path weights are restricted to be integers. It is a textbook result [1] that if G is acyclic (a DAG) a decomposition using no more than $m = |E|$ paths always exists. However, MFD is strongly NP-hard [25], even on DAGs, and



© Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumej, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams;

licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 31; pp. 31:1–31:14

Leibniz International Proceedings in Informatics



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

even when the flow values come only from $\{1, 2, 4\}$ [12]. Recent work has shown that the problem is FPT in the size of the minimum decomposition [14] and that it can be formulated as an ILP of quadratic size [7].

While difficult to solve, MFD is a key step in many applications. For example, MFD on DAGs is used to reconstruct biological sequences such as RNA transcripts [18, 23, 11, 3, 22, 26] and viral strains [2]. MFD can also be used to model problems in networking [25, 12, 15] and transportation planning [16], although in some of these applications there may be cycles in the input. Despite the ubiquity of the MFD problem, the gap in our knowledge about the approximability of MFD is large, even when restricting to DAGs. It is known [12] that MFD (even on DAGs) is APX-hard (i.e., there is some $\epsilon > 0$ such that it is NP-hard to approximate within a $(1 + \epsilon)$ factor), so in particular, MFD does not admit a PTAS, unless $P=NP$. Furthermore, it can be approximated with a factor of $\lambda^{\log \|X\|} \log \|X\|$ [15], where λ is the length of the longest source-to-sink path and $\|X\|$ is the largest flow value in the network. In this work, we attempt to fill in some of the gaps between these results.

A natural lower bound for the size of an MFD of a DAG is the size of a *minimum path cover* of the set of edges with non-zero flow (i.e., the minimum number of paths such that every such edge appears in *at least* one path) – this size is called the *width* of the network. This trivially holds because every flow decomposition is also such a path cover. These two notions are analogies of the more standard notions of path cover and width of the *node set*. The node-variants are classical concepts, with algorithmic results dating back to Dilworth and Fulkerson [8, 10]. Despite this, the width has not been given any attention in the MFD problem, and in particular it has never been used in approximation algorithms. In this paper, we show that the width can play a key role both in the analysis of popular heuristics, and in obtaining the first approximation algorithm for a natural variant of MFD.

We start with a relaxation of MFD in which flow decomposition may also use negative integer weights on flow paths, rather than strictly positive weights as has traditionally been considered [25, 12, 14]. An important observation that we leverage for this variant (unlike the positive-only version) is that “the width does not increase” as flow paths are chosen and removed. Using this, we give a $(\lceil \log \|X\| \rceil + 1)$ approximation algorithm for this variant. To differentiate both versions, we use $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$ throughout the paper. While $\text{MFD}_{\mathbb{Z}}$ is a natural version of the problem, to our knowledge it has not been previously considered in the MFD literature. However, it can also have natural applications, since by applying $\text{MFD}_{\mathbb{Z}}$ on the difference between two flows, one can minimally explain the *differences* between them, e.g. to explain the differences in RNA expression between two tissue samples with the fewest number of up/down regulated transcripts, which is often the goal of RNA sequencing experiments [21]. Our approximation follows a *power-of-two* approach where the weights of the paths chosen are (positive or negative) powers of two. More specifically, observe that if all flow values are even, then one can divide them by 2 and obtain a flow X with smaller $\|X\|$ whose decomposition can be transformed back into a decomposition of X . In order to obtain such an even flow, we prove a basic property that can be of independent interest: given any integer flow X , there exists a *unitary* flow (its values are 0, +1, or -1) Y , such that $X + Y$ is even on every edge (Lemma 5). In addition, given a unitary flow Y , we show that Y can be decomposed into k paths of weight +1 or -1 , such that k is at most the width of the graph (Corollary 8). We obtain the $(\lceil \log \|X\| \rceil + 1)$ approximation ratio (Theorem 11) by iteratively removing the unitary flow, dividing all flow values by 2, and preprocessing the graph so that its width is a lower bound on the size of the $\text{MFD}_{\mathbb{Z}}$.

In Section 4 we consider connections between the width and a popular heuristic algorithm for $\text{MFD}_{\mathbb{N}}$ which we call *greedy-weight*¹ [25], which builds a flow decomposition by successively choosing the path that can carry the largest flow. Greedy-weight is commonly used in applications (see e.g., [23, 2, 18] among many), and it seems to be mentioned in nearly every publication addressing flow decomposition. First, on sparse graphs we improve (i.e., increase) the worst-case lower bound for the greedy-weight approximation factor from $\Omega(\sqrt{m})$ [12] to $\Omega(m/\log m)$, showing for the first time that greedy-weight can be exponentially worse than the optimum. For this we use a class of sparse graphs where the optimum flow decomposition has size $O(\log m)$ whereas the greedy-weight algorithm returns a solution of size $\Omega(m)$, only a constant factor away from the trivial decomposition. The key to this new bound is to design an input where the width increases exponentially when a path is greedily removed. We also show that the same bound also holds for other greedy heuristics choosing instead the longest or shortest paths. Second, we show that if the input satisfies the properties that its width does not increase as source-to-sink paths are removed (Property 15) and that it is possible to remove a path of large weight (Property 16), then greedy-weight is a $O(\log |X|)$ -approximation, where $|X|$ is the flow value (i.e., total flow out of s). A notable class of graphs satisfying these properties is the class of *series-parallel graphs*; see [9, 24] for fast recognition algorithms and pointers to other NP-hard problems that are easier on this class of graphs. Series-parallel graphs are also of great interest for network flow problems (see, e.g., [13, 4]).

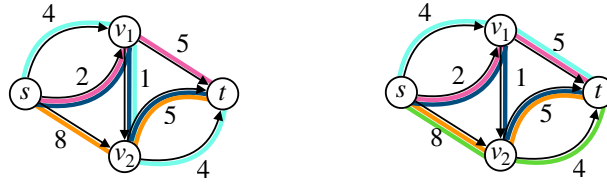
Finally, in Section 5 we consider a closely related problem, called *k-Flow Weight Assignment* [14]. In addition to the flow X , in this problem we are also given a set of k paths, and we need to decide if there is an assignment of weights to the paths such that they form a decomposition of X . If the weights belong to \mathbb{N} , this was shown to be NP-complete in [14]. In this work, we first observe that in the same way that allowing negative integer weights simplifies the approximability of MFD, allowing weights to belong to \mathbb{Z} fully changes the complexity of the k -Flow Assignment Problem, making it polynomial. This is due to the fact that the linear system defined by the given paths loses its only inequality of restricting the weights to positive integers. It thus transforms an ILP to a system of linear Diophantine equations, which can be solved in polynomial time (see e.g. [19]). Second, we consider a conjecture from [14] stating that if the weights belong to \mathbb{N} , and k is the size of a $\text{MFD}_{\mathbb{N}}$ for X , then the problem admits a unique solution (i.e., a unique assignment of weights to the given paths). If true, this would speed up the FPT algorithm of [14] for $\text{MFD}_{\mathbb{N}}$, because a step solving an ILP could be executed by solving a standard linear program returning a rational solution and checking if the (supposedly unique) solution to this system is integer. Moreover, the same conjecture (with the same implication) was also a motivation behind the greedy algorithm of [20] for $\text{MFD}_{\mathbb{N}}$. In this paper, we disprove the conjecture of [14], further corroborating the gap between $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$.

2 Preliminaries

We are given a directed graph $G = (V, E)$. Without loss of generality, we assume a unique source s and a unique sink t with no in-edges and no out-edges respectively; otherwise, the graph can be converted to such a graph by adding a pseudo source and sink and connecting them to all sources and sinks respectively with appropriately weighted edges. We also assume

¹ Previous work has consistently referred to this algorithm as *greedy-width*. To avoid confusion with the width of the graph, we introduce the name *greedy-weight* in this work.

31:4 Width Helps and Hinders Splitting Flows



(a) If paths and -

(b) With positive weights only, five paths are needed, since the edge (v_1, v_2) must be decomposed by a weight 1 path, leaving 4 edges that must be covered separately. The paths shown are one such decomposition.

■ **Figure 1** A positive flow admitting a decomposition into four paths only if negative weights are allowed.

that every node is on some s - t path. We use n and m to denote the number of nodes and edges of G , respectively. Additionally, we assume that G is a DAG throughout the paper. While the problem is also studied for graphs with cycles (see, e.g., [25, 12]), the task is still to decompose into simple paths, and so our inapproximability result on DAGs also applies for graphs with cycles. We call functions $X : E \rightarrow \mathbb{Y}$ *pseudo-flows*, where \mathbb{Y} is some set of allowed flow values (numbers). We treat pseudo-flows as vectors over E and use the notation $X + Y$ and aX to denote the (element-wise) sum of pseudo-flows and multiplication by a scalar, respectively. The numbers 0 and 1 also denote (depending on context) pseudo-flows that are 0 (resp. 1) everywhere. We write $X \leq Y$ (and similarly $<$) to mean $X(u, v) \leq Y(u, v)$ for every $(u, v) \in E$.

Given G , a *flow* is a pseudo-flow satisfying conservation of flow (incoming flow equal to outgoing flow) on internal nodes $V \setminus \{s, t\}$. It is known that the sum of two flows $X + Y$, the multiplication of a flow with a scalar aX , and the empty flow 0 are themselves flows. Let $|X|$ denote the total flow out of s (or into t) for flow X . Given an s - t path P , let P also denote the flow defined by setting 1 to every edge in P and 0 elsewhere. With these definitions, we are ready to formally define MFD.

► **Definition 1.** Given a flow X , a flow decomposition of (G, X) of size k is a family of s - t paths $\mathcal{P} = (P_1, \dots, P_k)$ with weights $(w_1, \dots, w_k) \in \mathbb{Y}^k$ such that $X = w_1P_1 + \dots + w_kP_k$.

► **Definition 2.** Given a flow X , let $\text{mfd}_{\mathbb{Y}}(G, X)$ be the smallest size of a flow decomposition of (G, X) with weights in \mathbb{Y} .

We omit \mathbb{Y} if it is clear from the context. We call the problem of producing a flow decomposition of (G, X) of minimum size the *minimum flow decomposition (MFD) problem*. In this paper, we study two integer versions of the problem, $\text{MFD}_{\mathbb{N}}$ ($0 \in \mathbb{N}$) and $\text{MFD}_{\mathbb{Z}}$. Note that the reduction showing $\text{MFD}_{\mathbb{N}}$ to be strongly NP-hard from [25] also holds for $\text{MFD}_{\mathbb{Z}}$. However, a positive flow may admit a decomposition using fewer paths if negative weights are allowed, as shown in Figure 1. We explore further differences between $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$ in Sections 3 and 5.

Let $\|X\| = \max_{(u,v) \in E} |X(u, v)|$ denote the infinity norm on flows. In particular, notice that if $\mathbb{Y} \subseteq \mathbb{Z}$, then $\|X\| \leq 1$ means that $X(u, v) \in \{0, \pm 1\}$ for every $(u, v) \in E$. Let $X \equiv_2 Y$ if X and Y have the same parity everywhere, i.e., for every $(u, v) \in E$, we have that $X(u, v)$ is odd iff $Y(u, v)$ is odd.

► **Definition 3.** Given $S \subseteq E$, we define $\text{width}_S(G)$ as the minimum number of s - t paths in G needed to cover all edges of S . If $S = E$ we just write $\text{width}(G)$.

Just like its more common node variant, $\text{width}(G)$ can be computed in $O(mn)$ time. As described by, e.g., [1, 6], this is done by reduction to a min-flow instance with demand one on every edge; the minimum flow of this instance is $\text{width}(G)$, and the flow can be found by reduction to a max-flow instance. Moreover, the problem can be relaxed to only require the coverage of $S \subseteq E$ and solved in the same running time by setting the demands only on the edges of S .

► **Lemma 4** ([1, 17]). *Let $G = (V, E)$ be a DAG, and $S \subseteq E$. A flow $C : E \rightarrow \mathbb{N}$ can be computed in $O(mn)$ time, such that $C(e) \geq 1$ for all $e \in S$ and $|C| = \text{width}_S(G)$.*

The flow C with total flow $\text{width}_S(G)$ suffices in this paper, and we do not need to calculate a path cover achieving that minimum. However, we note that it can be directly computed given the flow C . We can think of this path cover as a flow decomposition of C into $\text{width}_S(G)$ weight-one paths, which can be found by greedily removing such paths from C until it is completely decomposed. Since each path has no more than $n - 1$ edges and since $\text{width}_S(G) \leq m$, the overall runtime of finding the path cover is $O(mn)$.

3 Width helps solve $\text{MFD}_{\mathbb{Z}}$

The idea behind our approximation algorithm for $\text{MFD}_{\mathbb{Z}}$ is that a flow $X : E \rightarrow \mathbb{Z}$ on DAG G can always be decomposed into $(\lceil \log \|X\| \rceil + 1) \cdot \text{width}(G)$ paths. We show this using two key facts: first, that X can be decomposed into $(\lceil \log \|X\| \rceil + 1)$ flows with a particular structure, and, second, that each of these flows can be decomposed into $\text{width}(G)$ paths. A key step in proving both these facts is a subroutine which, given an input flow X , finds another flow Y with only values from $\{0, \pm 1\}$ (a *unitary* flow) that matches the parity of X on all the edges. Intuitively, given an input flow X , such a unitary flow Y can be added to X to “fix” its odd edges to be even, with only a small change to X .

► **Lemma 5.** *For any flow $X : E \rightarrow \mathbb{Z}$ on $G = (V, E)$, there exists a flow $Y : E \rightarrow \mathbb{Z}$ such that $X \equiv_2 Y$ and $\|Y\| \leq 1$.*

Proof. Consider the undirected graph $G_{\text{odd}} = (V, E_{\text{odd}})$ where $E_{\text{odd}} = \{\{u, v\} \mid (u, v) \in E \text{ and } X(u, v) \text{ is odd}\}$.

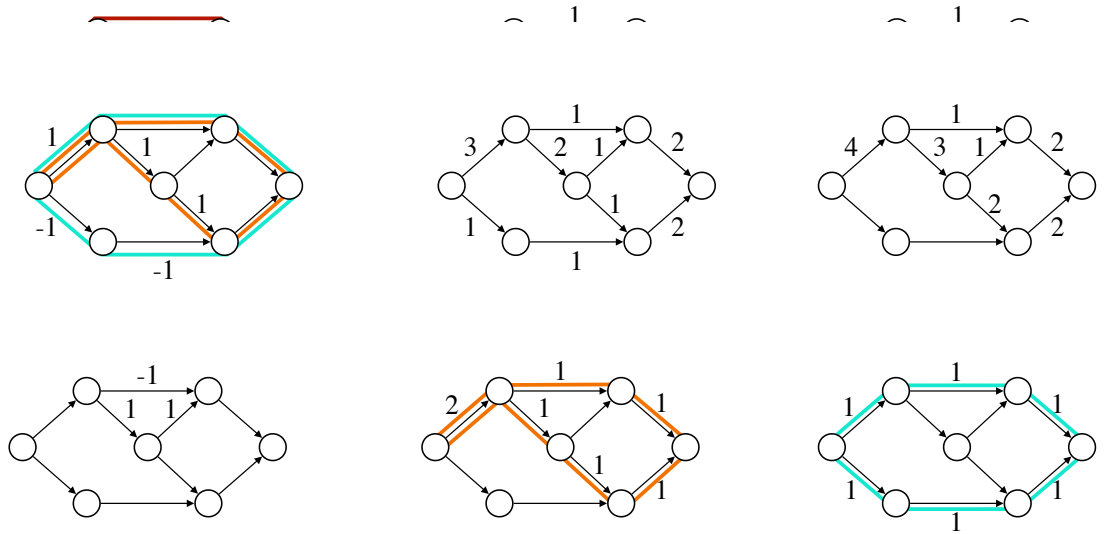
Notice that every node of G_{odd} , except possibly s and t , has even degree, due to conservation of flow. Thus, G_{odd} can be written as the edge-disjoint union of cycles and s - t paths. Assign an arbitrary orientation to each cycle and s - t path, and let E_{odd}^+ be the set of edges oriented in this way. Define

$$Y(u, v) = \begin{cases} +1 & \text{if } (u, v) \in E_{\text{odd}}^+ \\ -1 & \text{if } (v, u) \in E_{\text{odd}}^+ \\ 0 & \text{if } \{u, v\} \notin E_{\text{odd}} \end{cases}$$

Notice that Y is a flow decomposed as a sum of flows, each along one of the edge-disjoint cycles and s - t paths. Moreover, $X \equiv_2 Y$ and $\|Y\| \leq 1$ by construction. ◀

Repeatedly applying Lemma 5 and dividing the resulting even flow by 2, we obtain the the first key ingredient of the approach (proof in [5, Appendix B]).

► **Corollary 6.** *Any (non-zero) flow $X : E \rightarrow \mathbb{Z}$ can be written as $X = \sum_{i=0}^{\lceil \log \|X\| \rceil} 2^i \cdot Y_i$, where $Y_i : E \rightarrow \mathbb{Z}$ is a flow with $\|Y_i\| \leq 1$ for all i .*



(f) Flow $B = (C - D - X)/2$ and a decomposition of it into two paths of weight 1.

■ **Figure 2** Example of Lemma 7 and Corollary 8 applied to a unitary flow X on a graph G (for clarity, 0 flow values are not shown). Positive flows A and B can be constructed so that $|A| + |B| \leq \text{width}(G)$ holds. Flows A and B can be trivially decomposed into $|A|$ and $|B|$ paths, respectively. We obtain a decomposition of X by taking the paths of A with weight 1 and the paths of B with weight -1 .

The following result is the second key ingredient of our approach. It guarantees (together with Corollary 8) that any unitary flow can be decomposed into at most $\text{width}(G)$ paths of weight ± 1 (see Figure 2 for an example). This is by no means obvious since, among other problems, a unitary flow may contain positive and negative values which merge and cancel each other out (as in Figure 2a). The proof is based on another application of Lemma 5, along with some algebra on flows.

► **Lemma 7.** For any flow $X : E \rightarrow \mathbb{Z}$, $\|X\| \leq 1$, there exist flows $A, B : E \rightarrow \mathbb{Z}$ such that:

1. $A, B \geq 0$
2. $X = A - B$
3. $|A| + |B| \leq \text{width}(G)$

Proof. Take C such that $C \geq 1$ and $|C| = \text{width}(G)$, according to Lemma 4. Take D such that $D \equiv_2 X + C$ and $\|D\| \leq 1$, according to Lemma 5. Also, assume $|D| \geq 0$ without loss of generality (otherwise, take $-D$, which satisfies the same properties).

Since $D \equiv_2 X + C$, we have $C - D \pm X \equiv_2 0$. So we can take $A = (C - D + X)/2$ and $B = (C - D - X)/2$.

1. Notice that $C - D \pm X \geq C - 2$ since $\|D\|, \|X\| \leq 1$. So, $C - D \pm X \geq -1$, since $C \geq 1$. But $C - D \pm X \equiv_2 0$ so $C - D \pm X \geq 0$, whence $A, B \geq 0$.
2. $A - B = \frac{C - D + X}{2} - \frac{C - D - X}{2} = X$.
3. $|A| + |B| = |A + B| = \left| \frac{C - D + X}{2} + \frac{C - D - X}{2} \right| = |C - D| = |C| - |D| \leq |C|$ since $|D| \geq 0$, and $|C| = \text{width}(G)$. ◀

► **Corollary 8.** For any flow $X : E \rightarrow \mathbb{Z}$ with $\|X\| \leq 1$, there exist paths P_1, \dots, P_k with $k \leq \text{width}(G)$ such that $X = P_1 + \dots + P_\ell - P_{\ell+1} - \dots - P_k$ (for some $0 \leq \ell \leq k$).

Proof. Take A, B according to Lemma 7, with $A, B \geq 0$, $X = A - B$ and $|A| + |B| \leq \text{width}(G)$. Since $A, B \geq 0$, there exist paths $P_1, \dots, P_{|A|+|B|}$ such that $A = P_1 + \dots + P_{|A|}$ and $B = P_{|A|+1} + \dots + P_{|A|+|B|}$ [1]. Since $X = A - B$, we can write $X = P_1 + \dots + P_{|A|} - P_{|A|+1} - \dots - P_{|A|+|B|}$. Finally, recall that $|A| + |B| \leq \text{width}(G)$, concluding the proof. ◀

Finally, expressing any flow as a sum of at most $\lceil \log \|X\| \rceil + 1$ unitary flows (Corollary 6), and decomposing each unitary flow into at most $\text{width}(G)$ positive or negative paths (Corollary 8), we can decompose the flow into at most $\lceil \log \|X\| \rceil + 1$ paths whose weight are positive and negative powers of two.

► **Theorem 9.** *Given a DAG $G = (V, E)$, for any flow $X: E \rightarrow \mathbb{Z}$, there exist paths P_1, \dots, P_k and weights $\{w_1, \dots, w_k\} \subseteq \{\pm 2^i \mid i \in \mathbb{N}\}$, with $k \leq (\lceil \log \|X\| \rceil + 1) \cdot \text{width}(G)$ such that $X = w_1 P_1 + \dots + w_k P_k$.*

Proof. Combine Corollaries 6 and 8, getting

$$X = \sum_{i=0}^{\lceil \log \|X\| \rceil} 2^i \cdot Y_i = \sum_{i=0}^{\lceil \log \|X\| \rceil} 2^i \cdot (P_{\ell_i} + \dots + P_{\ell_i} - P_{\ell_i+1} - \dots - P_{k_i})$$

where $k_i \leq \text{width}(G)$. ◀

The proof of Theorem 9 suggests a straightforward algorithm for $\text{MFD}_{\mathbb{Z}}$, which we detail in Algorithm 2 and describe at a high level here. First, iteratively decompose X , yielding $\lceil \log \|X\| \rceil + 1$ unitary flows. Then use Lemma 7 to decompose each into $\text{width}(G)$ paths. However, as we explained at the beginning of this section, $\text{width}(G)$ is not a lower bound on $\text{MFD}_{\mathbb{Z}}$, and thus this approach does not directly derives an approximation. To overcome this issue, we instead find a flow decomposition of a spanning subgraph G' of G whose width lower bounds $\text{mfd}_{\mathbb{Z}}(G, X)$. Namely, we first find a minimum path cover flow in G of the subset S of edges with non-zero flow in $O(mn)$ time (according to Lemma 4), and then remove from G any edge not covered by the flow, obtaining G' . By construction, the size of this path cover is a lower bound of $\text{mfd}_{\mathbb{Z}}(G, X)$. Moreover, the size of this path cover is exactly $\text{width}(G')$, since every path cover of G' is also a path cover of S in G .

To prove the correctness of Algorithm 2, we first define a subroutine implementing Lemma 5.

► **Lemma 10.** *Algorithm 1 returns a unitary flow from input flow Y such that $X \equiv_2 Y$, as in Lemma 5, in $O(m)$ time.*

Proof. The correctness of the algorithm is given by Lemma 5. Finally, the first 3 subroutines as well as the entire **for**-loop takes $O(m)$ time. ◀

► **Theorem 11.** *Algorithm 2 is a $\log \lceil \|X\| \rceil + 1$ -approximation for $\text{MFD}_{\mathbb{Z}}$ with runtime $O(m(\log \|X\| \cdot \text{mfd}_{\mathbb{Z}}(G, X) + n)) = O(m^2 \log \|X\|)$.*

Proof. By Theorem 9 and our previous discussion, Algorithm 2 returns a flow decomposition for X with no more than $(\lceil \log \|X\| \rceil + 1) \cdot \text{width}(G') = (\lceil \log \|X\| \rceil + 1) \cdot \text{mfd}_{\mathbb{Z}}(G, X)$ paths. We analyze the runtime line by line. Lines 2 and 5 take $O(mn)$ time by Lemma 4. The call to Algorithm 1 on line 6 takes $O(m)$ time by Lemma 10, and checking the flow of D and flipping signs (if necessary) also takes $O(m)$ time. By Corollary 6, the while loop on line 8 executes at most $\log \lceil \|X\| \rceil + 1$ times, meaning that the entire execution takes $O(m \log \|X\|)$ time since line 9 takes $O(m)$ time by Lemma 10. Since there are at most $\log \lceil \|X\| \rceil + 1$ Y_i 's, the for loop on line 14 executes at most $\log \lceil \|X\| \rceil + 1$ times. Each

■ **Algorithm 1** Produces a unitary flow Y from input flow X such that $X \equiv_2 Y$, as in Lemma 5.

```

1: procedure UNITARY( $G, X$ )
2:    $E_{\text{odd}} \leftarrow$  odd edges of  $G$ , undirected
3:    $C \leftarrow$  a decomposition of  $G_{\text{odd}} = (V, E_{\text{odd}})$  into cycles, oriented arbitrarily
4:    $E_{\text{odd}}^+ \leftarrow$  directed edges of  $C$ 
5:   for  $(u, v) \in E$  do
6:     if  $(u, v) \in E_{\text{odd}}^+$  then
7:        $Y(u, v) \leftarrow +1$ 
8:     else if  $(v, u) \in E_{\text{odd}}^+$  then
9:        $Y(u, v) \leftarrow -1$ 
10:    else
11:       $Y(u, v) \leftarrow 0$ 
12:    end if
13:  end for
14:  return  $Y$ 
15: end procedure

```

execution of the for-loop naively finds $\text{width}(G')$ paths, each of which can be found in $O(m)$ time, so the whole loop takes $O(\log \|X\| \cdot \text{width}(G') \cdot m)$ time. Thus, the overall runtime is $O(m \log \|X\| \cdot \text{width}(G') + n) = O(m \log \|X\| \cdot \text{mfd}_{\mathbb{Z}}(G, X) + n)$. ◀

A theorem analogous to Theorem 9 for $\text{MFD}_{\mathbb{N}}$ is desirable, but cannot be achieved directly with the previous methods. Lemma 5 makes use of negative weights, and yields positive weights only if the flow graph solely consists of s - t paths. However, the approach can be adapted for $\text{MFD}_{\mathbb{N}}$ if the input flows are stable (Property 15), and if it is possible to “fix” the odd flows to be even with only $\text{width}(G)$ paths, which we leave as an open question.

4 Width matters for greedy approaches

Since the difference of two flows is still a flow, it is very natural to consider successively removing the most obvious type of flow – that is to say, paths – as an algorithmic strategy for $\text{MFD}_{\mathbb{N}}$. Indeed, the particular greedy path removal strategy of finding the *heaviest path* (*greedy-weight*) is commonly used as a heuristic in applications (e.g., [18, 2, 23, 12]) and it seems to be mentioned in nearly every paper addressing flow decomposition. More formally, a path P is said to *carry* flow p if $X(u, v) \geq p$ for all edges (u, v) of P . The *heaviest path* is an s - t path carrying the largest flow. Such a path can be easily found in linear time in the size of the DAG by dynamic programming (see, e.g., [25]).

4.1 Width hinders greedy on $\text{MFD}_{\mathbb{N}}$

We define a family of $\text{MFD}_{\mathbb{N}}$ instances $(G_{\ell}, X_{\ell, B})$, depending on two parameters $\ell \in \mathbb{N} \setminus \{0\}$ and $B \in \mathbb{N}$. The family is defined recursively on ℓ . The base case $(G_1, X_{1, B})$ for $\ell = 1$ is shown in Figure 3a. For $\ell > 1$, we build $(G_{\ell}, X_{\ell, B})$ from two disjoint copies of $(G_{\ell-1}, X_{\ell-1, B})$, by adding 5 extra edges and flow values as shown in Figure 3b. We call the edge connecting the two copies of $G_{\ell-1}$ a *central edge*. Edges whose flow value depends on B are called *backbone* edges, and they form a s - t path. Choosing $B = 2^{\ell+1}$, we show that the flow $X_{\ell, 2^{\ell+1}}$ can be decomposed using a number of paths linear in ℓ , thanks to the heavy backbone

■ **Algorithm 2** Finds the flow decomposition of Theorem 9.

```

1: procedure PATH-DECOMPOSITION( $G, X$ )
2:   Compute a minimum path cover flow of  $\{(u, v) \in E \mid X(u, v) \neq 0\}$   $\triangleright$  Lemma 4
3:   Remove from  $G$  any edge not covered by this path cover to obtain  $G'$ 
4:    $\mathcal{P} \leftarrow [], w \leftarrow []$   $\triangleright$  length-zero vectors
5:    $C \leftarrow$  flow of value  $\text{width}(G')$ ,  $C \geq 1$   $\triangleright$  Lemma 4
6:    $D \leftarrow \text{UNITARY}(G', C)$ ; if  $|D| < 0$  set  $D = -D$   $\triangleright$  Algorithm 1
7:    $i \leftarrow 0$ 
8:   while  $\|X\| > 1$  do
9:      $Y_i \leftarrow \text{UNITARY}(G', X)$   $\triangleright$  Algorithm 1
10:     $X \leftarrow (X - Y_i)/2$ 
11:     $i \leftarrow i + 1$ 
12:  end while
13:   $Y_i \leftarrow X$ 
14:  for  $j \in \{0, \dots, i\}$  s.t.  $Y_j \neq 0$  do
15:     $A \leftarrow C - D + Y_j$ ,  $B \leftarrow C - D - Y_j$ 
16:    Naively decompose  $A$  into  $|A|$  paths and  $B$  into  $|B|$  paths; concatenate  $A, B$  to  $\mathcal{P}$ 
17:    Concatenate  $|A|$  copies of  $2^j$  and  $|B|$  copies of  $-2^j$  to  $w$ 
18:  end for
19:  return  $(\mathcal{P}, w)$ 
20: end procedure

```

edges, whereas the greedy-weight algorithm fully saturates the central edges with its first path and is left with a remaining flow requiring $2^{\ell+1}$ paths to be decomposed (proofs in [5, Appendix B]).

► **Lemma 12.** *Let G_ℓ with flow $X_{\ell, 2^{\ell+1}}$ be constructed as described before. Greedy-weight uses $1 + 2^{\ell+1}$ paths to decompose $X_{\ell, 2^{\ell+1}}$.*

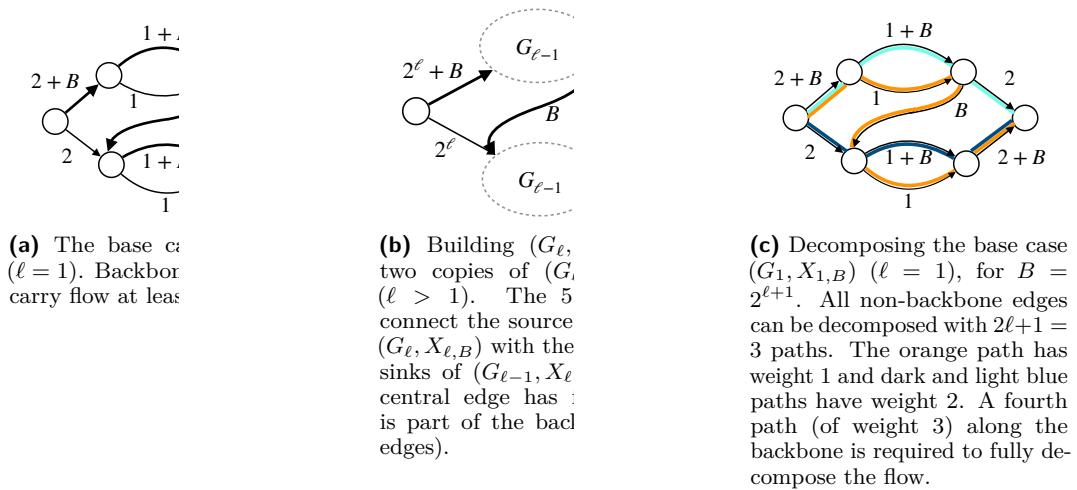
► **Lemma 13.** *Let G_ℓ with flow $X_{\ell, 2^{\ell+1}}$ be constructed as described before. It is possible to decompose $X_{\ell, 2^{\ell+1}}$ using $2\ell + 2$ paths.*

► **Theorem 14.** *The approximation ratio for greedy-weight on $\text{MFD}_{\mathbb{N}}$ is $\Omega(m/\log m)$ for sparse graphs, in the worst case.*

Proof. By Lemmas 12 and 13, greedy-weight uses $\Theta(2^\ell)$ paths to decompose the flow $X_{\ell, 2^{\ell+1}}$ described above, whereas it is possible to decompose the flow with only $\Theta(\ell)$ paths. It can be easily verified by induction that the number of edges of G_ℓ is $7 \cdot 2^\ell - 5$. So the ratio between greedy-weight and optimal for this instance is $\Omega(\frac{m}{\log m})$. ◀

While greedy-weight is most commonly used in applications, the approach was first presented as part of a general framework [25]: pick any optimality criteria for s - t paths that is *saturating* (i.e., fully decomposes at least one edge), and successively remove optimal paths. Since each path is saturating, the algorithm must decompose the flow in m or fewer paths. Another optimality criterion sometimes used in DNA assembly (e.g., in vg-flow [2]) is the *longest path* (with its maximum possible flow so that it is saturating). To adapt our construction of $(G_\ell, X_{\ell, 2^{\ell+1}})$ so that this approach has the same approximation ratio, consider $(G_\ell^*, X_{\ell, 2^{\ell+1}}^*)$, constructed as in $(G_\ell, X_{\ell, 2^{\ell+1}})$ except that we replace every backbone edge (u, v) with two edges, (u, w) and (w, v) . See [5, Figure 6] for an example. Then a path

31:10 Width Helps and Hinders Splitting Flows



■ **Figure 3** Construction for $(G_\ell, X_{\ell,B})$. Setting $B = 2^{\ell+1}$ gives MFD instances where greedy-weight uses $\Theta(\frac{m}{\log m})$ times more paths than optimal to decompose the flow.

along the backbone edges will be the longest from s to t and the above asymptotic analysis holds, since we no more than doubled the number of edges. Yet another optimality criterion, studied in [12] for its application to network routing, is the shortest path (again with its maximum possible flow). $(G_\ell^*, X_{\ell,2^{\ell+1}}^*)$ will also force this approach to take an exponential number of paths, since first the algorithm will take all $2^{\ell+1}$ weight-1 edges with $2^{\ell+1}$ different paths.

4.2 Greedy approximation for series-parallel graphs

As exploited in Section 4.1, one sticking point for greedy path removal algorithms is the fact that the width of a graph can increase after an edge is fully decomposed. We now show that if, in a particular instance, a graph does not increase its width during the execution of the algorithm, and greedy-weight can decompose “enough” flow at each step, then greedy-weight is a $O(\log |X|)$ -approximation for $\text{MFD}_{\mathbb{N}}$.

If G is a directed graph and $X \geq 0$ a flow on G , we write $G|_X$ (G restricted to X) to mean the spanning subgraph of G made up of the edges $e \in E$ such that $X(e) \neq 0$. Conversely, if S is a subgraph of G , we write $X|_S$ (X restricted to S) to mean the pseudo-flow X only on the edges of S . In the case of $\text{MFD}_{\mathbb{N}}$, once an edge is fully decomposed, it cannot be used in future paths, possibly increasing the width of the graph that can be used to decompose the rest of the flow and sometimes triggering an increase of the size of a minimum flow decomposition as well. We call a graph *stable* if it does not have this issue.

► **Property 15** (Stable graph). We say that G is *stable* if, for any non-negative flow X on G , it holds that $\text{width}(G|_X) \leq \text{width}(G)$.

Many useful $\text{MFD}_{\mathbb{N}}$ instances do in fact satisfy Property 15. For example, the first proof of MFD 's NP-hardness [25] was a reduction to a very simple graph of this form; this means that $\text{MFD}_{\mathbb{N}}$ restricted to stable graphs is also NP-hard.

The second property that we need is that there is always, during the execution of the algorithm, a path carrying “enough” flow from s to t .

► **Property 16** (Paths of large weight). We say that G has *paths of large weight* if, for any flow $X \geq 0$ on G , there exists an s - t path in $G|_X$ carrying at least $|X|/\text{width}(G)$ flow.

Note that this property does not hold in general; see [5, Figure 7].

► **Lemma 17.** *Let $G = (V, E)$ be a graph, $\text{width}(G) \geq 2$, satisfying Properties 15 and 16. Greedy-weight uses at most $\lfloor \log |X| / \log \frac{\text{width}(G)}{\text{width}(G)-1} \rfloor + 1$ paths to decompose any flow $X : E \rightarrow \mathbb{N}$.*

Proof. Let $b = \text{width}(G)$. Since G satisfies Properties 15 and 16, greedy-weight removes a path of weight at least $|X'|/b$ at every step, where X' is the remaining flow of the corresponding step. As such, after c steps $|X'| \leq |X| \left(\frac{b-1}{b}\right)^c$. If $|X| \left(\frac{b-1}{b}\right)^c < 1$, then $|X'| = 0$, since $|X|$ and the weights of the removed paths belong to \mathbb{N} . Solving for c we obtain $c > \log |X| / \log \frac{b}{b-1}$. Therefore, greedy-weight takes (uses) at most $c = \lfloor \log |X| / \log \frac{b}{b-1} \rfloor + 1$ steps (paths). ◀

► **Theorem 18.** *Let $G = (V, E)$ be a graph satisfying Properties 15 and 16 and $X : E \rightarrow \mathbb{N}$ a flow. Greedy-weight is a $O(\log |X|)$ -approximation for $\text{MFD}_{\mathbb{N}}$ on (G, X) .*

Proof. Assume $X > 0$ (otherwise, replace G with $G|_X$). Thus, $b = \text{width}(G) \leq \text{mfd}_{\mathbb{N}}(G, X)$, since any flow-decomposition of X induces a path cover of E . If $b \leq 1$ greedy-weight finds an optimal solution. Otherwise $b \geq 2$, and Lemma 17 implies that greedy-weight is a $O\left(\frac{\log |X|}{b \log \frac{b}{b-1}}\right) = O(\log |X|)$ -approximation for $\text{MFD}_{\mathbb{N}}$ ($b \log \frac{b}{b-1} = O(1)$ for $b \geq 2$). ◀

Finally, we define series-parallel graphs, and apply Theorem 18 to them, by proving (in [5, Appendix B]) that they satisfy Properties 15 and 16.

► **Definition 19** (Series-parallel graph [9]). *A graph is a two-terminal series-parallel (series-parallel for short) graph with terminal nodes s and t if:*

- *it consists of a single edge directed from s to t , and no other nodes, or*
- *it can be obtained from two (smaller) two-terminal series-parallel graphs G_1 and G_2 , with terminal nodes s_1, t_1 , and s_2, t_2 , respectively, by either*
 - *identifying $s = s_1 = s_2$ and $t = t_1 = t_2$ (parallel composition of G_1 and G_2), or*
 - *identifying $s = s_1, t_1 = s_2$, and $t = t_2$ (series composition of G_1 and G_2).*

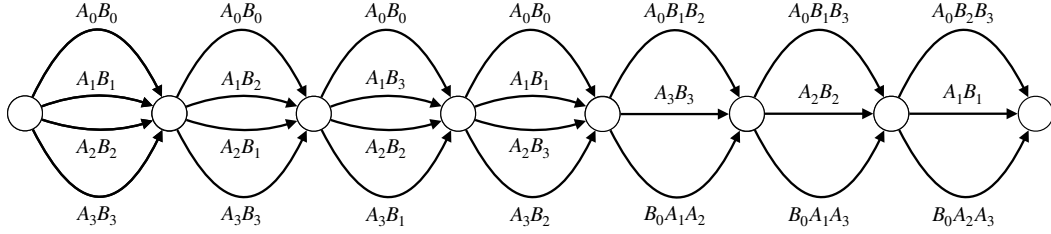
► **Corollary 20.** *Greedy-weight is a $O(\log |X|)$ -approximation for $\text{MFD}_{\mathbb{N}}$ on series-parallel graphs.*

5 Solving the k -Flow Weight Assignment Problem

In this section, we consider a restriction of MFD from [14] (see Figure 4 for an example).

► **Definition 21** (k -Flow Weight Assignment). *Given a flow $X : E \rightarrow \mathbb{Y}$ on a graph $G = (V, E)$ and a set of s - t paths $\{P_1, \dots, P_k\}$, the problem of finding an assignment of weights to the paths, such that they form a flow decomposition of (G, X) , is called k -Flow Weight Assignment (k -FWA). We write $k\text{-FWA}_{\mathbb{Y}}$ if we require the path weights to belong to \mathbb{Y} .*

Given k s - t paths, k -FWA can be solved by a linear system defined by $Lw = X$, where $X_j \in \mathbb{Y}$ is equal to the flow $X(e_j)$ of the edge e_j (we identify flows $X : E \rightarrow \mathbb{Y}$ with vectors $X \in \mathbb{Y}^m$) and L is the $m \times k$ 0/1 matrix with $L_{i,j} = 1$ if and only if path P_j crosses edge e_i . The resulting solution $w \in \mathbb{Y}^k$ is the weight assignment to each path. For a flow graph (G, X) , we denote by $L_{\mathbb{Y}} = L_{\mathbb{Y}}(P_1, \dots, P_k) = \{w \in \mathbb{Y}^k | X = \sum_{j=1}^k P_j w_j\}$ the linear system corresponding to the paths P_1, \dots, P_k .



■ **Figure 4** Paths A_i and B_i ($i \in \{0, 1, 2, 3\}$), each edge being labeled with the paths it appears in. Assign to each path A_i weight a_i , and to each path B_i weight b_i , such that $a_0 = b_0 = 3$, and $a_i = 6^{2i} + 1$ and $b_i = 6^{2i+1} + 5$ for $i = 1, 2, 3$. Define the flow X on G as $X = \sum_{i=0}^3 a_i A_i + \sum_{i=0}^3 b_i B_i$. Note that these weights are a solution of k -FWA $_{\mathbb{N}}$ on input (G, X) with given paths A_i, B_i ($i \in \{0, 1, 2, 3\}$).

We shortly discuss how to solve k -FWA $_{\mathbb{Z}}$. It is possible to find an integer-weighted solution or decide that it does not exist in polynomial time. The reason for this is that, having no non-negativity constraint, the linear system defined by the paths is a system of linear Diophantine equations. It is well known that integer solutions to such systems can be found in polynomial time; see, e.g., [19, Chapter 5].

On the other hand, solving k -FWA $_{\mathbb{N}}$ turns out to be more difficult. Here, the linear system contains the inequality $w \geq 0$. In fact, it was shown [14] that k -FWA $_{\mathbb{N}}$ is NP-hard. The program Toboggan [14] implements a linear FPT algorithm for MFD $_{\mathbb{N}}$. One step of the algorithm is to solve k -FWA $_{\mathbb{N}}$ using an ILP [14]. The authors state the following conjecture.

► **Conjecture 22** ([14]). *If (P_1, \dots, P_k) are the paths of a minimum flow decomposition of (G, X) , then the linear system $L_{\mathbb{N}}(P_1, \dots, P_k)$ has full rank k .*

As mentioned in the introduction, if the conjecture turned out to be true, then Toboggan could avoid resorting to solving an ILP, since just solving the standard linear system at hand would return its unique solution. As observed by the authors, this would decrease the asymptotic worst case upper bound of Toboggan.

We show that this conjecture is false using a counterexample. Consider the input for k -FWA $_{\mathbb{N}}$ from Figure 4 and the solution therein. We now give another solution for k -FWA $_{\mathbb{N}}$ on this input, namely the following path weights: $a_0 = 5, b_0 = 1$, and $a_i = 6^{2i} + 2, b_i = 6^{2i+1} + 4$, for $i = 1, 2, 3$. One can easily verify that this is another solution to k -FWA $_{\mathbb{N}}$ on the input in Figure 4, thus proving that the rank of the corresponding linear system is strictly less than 8.

To disprove Conjecture 22, it remains to show that any flow decomposition contains at least 8 paths. Due to the technicality of this proof (and its exhaustive case-by-case analysis), in this paper we only explain the intuition behind the construction in Figure 4 and behind the correctness proof. However, as an additional check we also ran both Toboggan [14] and a recently developed ILP solver for MFD $_{\mathbb{N}}$ [7] on this instance, both returning $\text{mfd}_{\mathbb{N}}(G, X) = 8$.

The intuition is as follows. The graph can be divided into two parts: the graph induced by the first 5 vertices in topological order (left part) and the one induced by the last 5 (right part). The exponential growth of the paths A_i and B_i for growing i , together with the different permutations of the paired labels $A_i B_j$ on the left part, fix the choice of the paths A_i and B_i for $i = 1, 2, 3$. This allows us to interpret flow decompositions of less than 8 paths as decompositions with 8 paths, where either A_0 or B_0 carries weight 0. Consider a flow decomposition where we assign two paths of weights λ_1 and λ_2 on the edges labeled $A_0 B_0$. For any $\delta \geq 0$, $(\lambda_1 - \delta) + (\lambda_2 + \delta) = a_0 + b_0$ and equivalently for all other edges on the left part. If we decrease λ_1 by some $\delta > 0$, the weights of B_1 and B_2 each increase by $\delta/2$. And thus, δ must be even. Due to the parity of a_0 and b_0 , they can never reach 0.

6 Conclusions

In this paper we have shown for the first time that width, a natural lower bound for MFD, is also useful when investigating its approximability. On the one hand, using width is a key insight in understanding where greedy path removal heuristics fail. On the other hand, graphs where width is well-behaved (e.g., series-parallel graphs) have a guaranteed approximation factor. Moreover, when combined with parity arguments, i.e., about parity fixing unitary flows, and a width-sized decomposition of such flows, it guarantees an even better approximation factor for $\text{MFD}_{\mathbb{Z}}$ for all DAGs. Finally, we have corroborated the complexity gap between the positive integer and the full integer case by disproving a conjecture from [14] (also motivating the heuristic in [20]), which would have had sped up their FPT algorithm for $\text{MFD}_{\mathbb{N}}$.

Our results open up new avenues for further research on MFD. For example, can the width help find larger classes of graphs for which some greedy path removal (or even some sort of greedy *path cover* removal) algorithms have a guaranteed approximation factor? Can we get $\Omega(n)$ worst case approximation ratio of greedy-weight for dense graphs without parallel edges? Can the power-of-two decomposition approach be applied with other factors besides two? Can better path cover-like lower bounds help (e.g., path covers which cannot use an edge more times than its flow value, also computable in polynomial time)? How do our algorithms perform in practice?

References


- 1 Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows: Theory, algorithms and applications. *New Jersey: Prentice-Hall*, 1993.
- 2 Jasmijn A Baaijens, Leen Stougie, and Alexander Schönhuth. Strain-aware assembly of genomes from mixed samples using flow variation graphs. In *International Conference on Research in Computational Molecular Biology*, pages 221–222. Springer, 2020.
- 3 Elsa Bernard, Laurent Jacob, Julien Mairal, and Jean-Philippe Vert. Efficient RNA isoform identification and quantification from RNA-Seq data with network flows. *Bioinformatics*, 30(17):2447–2455, 2014.
- 4 Dimitris Bertsimas, Ebrahim Nasrabadi, and Sebastian Stiller. Robust and adaptive network flows. *Operations Research*, 61(5):1218–1242, 2013. doi:10.1287/opre.2013.1200.
- 5 Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams. Width helps and hinders splitting flows. *CoRR*, 2022. doi:10.48550/ARXIV.2207.02136.
- 6 Manuel Cáceres, Massimo Cairo, Brendan Mumey, Romeo Rizzi, and Alexandru I. Tomescu. Sparsifying, shrinking and splicing for minimum path cover in parameterized linear time. In *SODA 2022 - ACM-SIAM Symposium on Discrete Algorithms*, pages 359–376, 2022. doi:10.1137/1.9781611977073.18.
- 7 Fernando H. C. Dias, Lucia Williams, Brendan Mumey, and Alexandru I. Tomescu. Fast, Flexible, and Exact Minimum Flow Decompositions via ILP. *arXiv*, arXiv:2201.10923, 2022. To appear in the Proceedings of RECOMB 2022 – 26th Annual International Conference on Research in Computational Molecular Biology. arXiv:2201.10923.
- 8 Robert P Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950. URL: <http://www.jstor.org/stable/1969503>.
- 9 David Eppstein. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41–55, 1992. doi:10.1016/0890-5401(92)90041-D.
- 10 Delbert R Fulkerson. Note on Dilworth’s decomposition theorem for partially ordered sets. *Proceedings of the American Mathematical Society*, 7(4):701–702, 1956.
- 11 Thomas Gatter and Peter F Stadler. Ryūtō: network-flow based transcriptome reconstruction. *BMC bioinformatics*, 20(1):1–14, 2019.

- 12 Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michal Segalov. How to split a flow? In *2012 Proceedings IEEE INFOCOM*, pages 828–836. IEEE, 2012.
- 13 A. Jain and N. Chandrasekharan. An efficient parallel algorithm for min-cost flow on directed series-parallel networks. In *Proceedings Seventh International Parallel Processing Symposium*, pages 188–192, 1993. doi:10.1109/IPPS.1993.262879.
- 14 Kyle Kloster, Philipp Kunke, Michael P O’Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. A practical fpt algorithm for flow decomposition and transcript assembly. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–86. SIAM, 2018.
- 15 Brendan Mumey, Samareh Shahmohammadi, Kathryn McManus, and Sean Yaw. Parity balancing path flow decomposition and routing. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2015.
- 16 Nils Olsen, Natalia Kliewer, and Lena Wolbeck. A study on flow decomposition methods for scheduling of electric buses in public transport based on aggregated time-space network models. *Central European Journal of Operations Research*, 2020. doi:10.1007/s10100-020-00705-6.
- 17 James B. Orlin. Max flows in $O(nm)$ time, or better. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774. ACM, 2013. doi:10.1145/2488608.2488705.
- 18 Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nature biotechnology*, 33(3):290–295, 2015.
- 19 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., 1986.
- 20 Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM transactions on computational biology and bioinformatics*, 16(2):658–670, 2017.
- 21 Mingxiang Teng, Michael I Love, Carrie A Davis, Sarah Djebali, Alexander Dobin, Brenton R Graveley, Sheng Li, Christopher E Mason, Sara Olson, Dmitri Pervouchine, et al. A benchmark for rna-seq quantification pipelines. *Genome biology*, 17(1):1–12, 2016.
- 22 Alexandru I Tomescu, Travis Gagie, Alexandru Popa, Romeo Rizzi, Anna Kuosmanen, and Veli Mäkinen. Explaining a weighted DAG with few paths for solving genome-guided multi-assembly. *IEEE/ACM transactions on computational biology and bioinformatics*, 12(6):1345–1354, 2015.
- 23 Alexandru I Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. In *BMC bioinformatics*, volume 14, pages S15:1–S15:10. Springer, 2013.
- 24 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982. doi:10.1137/0211023.
- 25 Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008.
- 26 Lucia Williams, Gillian Reynolds, and Brendan Mumey. RNA Transcript Assembly Using Inexact Flows. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1907–1914. IEEE, 2019.

Counting Simplices in Hypergraph Streams

Amit Chakrabarti   

Dartmouth College, Hanover, NH, USA

Themistoklis Haris 

Dartmouth College, Hanover, NH, USA

Abstract

We consider the problem of space-efficiently estimating the number of *simplices* in a hypergraph stream. This is the most natural hypergraph generalization of the highly-studied problem of estimating the number of *triangles* in a graph stream. Our input is a k -uniform hypergraph H with n vertices and m hyperedges, each hyperedge being a k -sized subset of vertices. A k -simplex in H is a subhypergraph on $k + 1$ vertices X such that all $k + 1$ possible hyperedges among X exist in H . The goal is to process the hyperedges of H , which arrive in an arbitrary order as a data stream, and compute a good estimate of $T_k(H)$, the number of k -simplices in H .

We design a suite of algorithms for this problem. As with triangle-counting in graphs (which is the special case $k = 2$), sublinear space is achievable but only under a promise of the form $T_k(H) \geq T$. Under such a promise, our algorithms use at most four passes and together imply a space bound of

$$O\left(\varepsilon^{-2} \log \delta^{-1} \text{polylog } n \cdot \min\left\{\frac{m^{1+1/k}}{T}, \frac{m}{T^{2/(k+1)}}\right\}\right)$$

for each fixed $k \geq 3$, in order to guarantee an estimate within $(1 \pm \varepsilon)T_k(H)$ with probability $\geq 1 - \delta$. We also give a simpler 1-pass algorithm that achieves $O\left(\varepsilon^{-2} \log \delta^{-1} \log n \cdot (m/T) \left(\Delta_E + \Delta_V^{1-1/k}\right)\right)$ space, where Δ_E (respectively, Δ_V) denotes the maximum number of k -simplices that share a hyperedge (respectively, a vertex), which generalizes a previous result for the $k = 2$ case. We complement these algorithmic results with space lower bounds of the form $\Omega(\varepsilon^{-2})$, $\Omega(m^{1+1/k}/T)$, $\Omega(m/T^{1-1/k})$ and $\Omega(m\Delta_V^{1/k}/T)$ for multi-pass algorithms and $\Omega(m\Delta_E/T)$ for 1-pass algorithms, which show that some of the dependencies on parameters in our upper bounds are nearly tight. Our techniques extend and generalize several different ideas previously developed for triangle counting in graphs, using appropriate innovations to handle the more complicated combinatorics of hypergraphs.

2012 ACM Subject Classification Theory of computation \rightarrow Sketching and sampling

Keywords and phrases data streaming, graph algorithms, hypergraphs, sub-linear algorithms, triangle counting

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.32

Related Version *Full Version*: <https://arxiv.org/abs/2112.11016>

Funding This work was supported in part by NSF under Award CCF-1907738.

1 Introduction

Estimating the number of triangles in a massive input graph is a fundamental algorithmic problem that has attracted over two decades of intense research [1, 3, 22, 8, 40, 41, 28, 35, 6, 36, 21, 7, 32, 5, 13, 25, 20]. It is easy to see why. On the one hand, the problem arises in applications where a complex real-world network is naturally modeled as a graph and the number of triangles is a crucial statistic of the network. Such applications are found in many different domains, such as social networks [11, 29, 33], the web graph [4, 44], and biological networks [38]; see [41] for a more detailed discussion of such applications. On the other hand, a triangle is perhaps *the* most basic nontrivial pattern in a graph and as such, triangle counting is a problem with a rich theory and connections to many areas within computer science [1, 15, 18, 34, 14] and combinatorics [31, 17, 27, 37].



© Amit Chakrabarti and Themistoklis Haris;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 32; pp. 32:1–32:19
Leibniz International Proceedings in Informatics



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

In this work, we study the natural generalization of this problem to massive *hypergraphs*. Just as graphs model pairwise interactions between entities in a network, hypergraphs model higher arity interactions. For instance, in an academic collaboration network with researchers being the vertices, it would be natural to model coauthorships on research papers and articles using *hyperedges*, each of which can be incident to more than two vertices. Just as we may use triangle counts to study clustering behaviors in graphs or even in different portions of a single graph, we may analyze higher-order clustering behaviors in the 3-uniform hypergraph H formed by all three-way coauthorships by counting 3-*simplices* in H . A 3-simplex on four vertices $\{u, v, w, x\}$ is the structure formed by the hyperedges $\{uvw, uvx, uwx, vwx\}$: it is the natural 3-dimensional analog of a triangle in an ordinary graph.

1.1 Our Results

We design several algorithms for space-efficiently estimating the number of k -simplices in a k -uniform hypergraph H given as a stream of hyperedges. A k -*simplex* is a complete k -uniform hypergraph on $k + 1$ vertices. The special case $k = 2$ is the triangle counting problem which, as noted, is intensely investigated. Indeed, even in this setting of streaming algorithms, triangle counting is highly studied, with new algorithmic techniques being developed as recently as 2021 [20]. There is also a body of work on generalizing these results to the problem of estimating the number of occurrences of patterns (a.k.a. *motifs*) more complicated than triangles, e.g., fixed-size cliques and cycles [30, 26, 36, 5, 24, 42]. Our work adds to this literature by generalizing in a different direction: we generalize the class of *inputs* from graphs to hypergraphs and focus on counting the simplest nontrivial symmetric motifs: k -simplices.

Our algorithms provide optimal space bounds (up to log factors) in certain parameter regimes; we prove this optimality by giving a set of matching lower bounds. In certain other parameter regimes, there remains a gap between our best upper bounds and lower bounds, which immediately provides a goal for future work on this problem. Below are informal statements of our major algorithmic results.

► **Theorem 1** (Upper bounds; informal). *Let H be an n -vertex k -uniform hypergraph, presented as a stream of m hyperedges, that is promised to contain T or more k -simplices. Suppose that each hyperedge is contained in at most Δ_E such simplices and each vertex is contained in at most Δ_V of them. Then, there are algorithms for $(1 \pm \varepsilon)$ -estimating $T_k(H)$, the number of k -simplices in H , with the following guarantees:*

(1.1) *a 4-pass algorithm using $\tilde{O}(m^{1+1/k}/T)$ space;*

(1.2) *a 2-pass algorithm using $\tilde{O}(m/T^{2/(k+1)})$ space, provided $k \geq 3$; and*

(1.3) *a 1-pass algorithm using $\tilde{O}(m(\Delta_E + \Delta_V^{1-1/k})/T)$ space.*

*Each of these algorithms is randomized and fails with probability at most δ .*¹

Formal versions of these results appear as Theorem 14 in Section 3, Theorem 16 in Section 4.2, and in the full paper [9], respectively. Along the way, we also obtain some other algorithmic results that, despite being dominated by the algorithms behind Theorem 1, are technically interesting and possibly useful in future work. These are briefly described at the start of Sections 3 and 4 with full details appearing in the full paper [9]. As we shall see, we take several ideas from triangle-counting algorithms as inspiration, but the “correct” way to extend these ideas to hypergraphs is far from obvious. Indeed, some of the more “obvious” extensions lead to the less-than-best algorithms hinted at above.

¹ Throughout this paper, the notation $\tilde{O}(\cdot)$ hides factors of $O(\varepsilon^{-2} \log \delta^{-1} \text{polylog } n)$ and, in the context of k -uniform hypergraphs, treats k as a constant. Also, “log” with an unspecified base means \log_2 .

Next, we informally state our lower bound results. These are not the main technical contributions of this paper, but they play the important role of clarifying where our algorithms are optimal and where there might be room for improvement. As in Theorem 1, we denote the number of k -simplices in H by $T_k(H)$.

► **Theorem 2** (Lower bounds; informal). *Let $k, n, m, H, \Delta_E, \Delta_V$ be as above. Suppose an algorithm makes p passes over a stream of hyperedges of H , using at most S bits of working memory, and distinguishes between the cases $T_k(H) = 0$ and $T_k(H) \geq T$ with probability at least $2/3$. Then the following lower bounds apply.*

(2.1) *With $T = 1$ and $p = O(1)$, sublinear space is impossible: we must have $S = \Omega(n^k)$.*

(2.2) *With $p = 1$, we must have $S = \Omega(m\Delta_E/T)$.*

(2.3) *With $p = O(1)$, we need $S = \Omega(\min\{m^{1+1/k}/T, m/T^{1-1/k}\})$, and $S = \Omega(m\Delta_V^{1/k}/T)$.*

[(2.1)] *If, instead, the streaming algorithm distinguishes between the cases $T_k(H) < (1 - \varepsilon)T$ and $T_k(H) > (1 + \varepsilon)T$, then the following lower bound applies.*

(2.4) *With $p = O(1)$, we must have $S = \Omega(\varepsilon^{-2})$.*

The full version of the paper [9] further discusses and proves these lower bounds.

1.2 Closely Related Previous Work

To the best of our knowledge, this is the first work to study the simplex counting problem in hypergraphs in the setting just described (the work of [23] on counting general hypergraph patterns is not closely related; see Section 1.3). We now summarize some highlights of previous work on the triangle counting problem (in graphs), with a focus on streaming algorithms, so as to provide context for our contributions.

Suppose an n -vertex m -edge graph G is given as a stream of edges and we wish to estimate the number of triangles, $T_3(G)$. It is not hard to show that, absent any promises on the structure of G , this problem requires $\Omega(n^2)$ space, even with multiple passes, thus precluding a sublinear-space solution. Therefore, all work in this area seeks bounds under a promise that $T_3(G) \geq T$, for some nontrivial threshold T . Intuitively, the larger this threshold, the easier the problem, so we expect the space complexity to decrease. The earliest nontrivial streaming solution [3] reduced triangle counting to a combination of ℓ_0 , ℓ_1 , and ℓ_2 estimation and achieved $\tilde{O}((mn/T)^2)$ space by using suitable linear sketches. Almost all algorithms developed since then have instead used some sort of *sampling* to extract a small portion of G , perform some computation on this sample, and then extrapolate to estimate $T_3(G)$.

Over the years, a number of different sampling strategies have been developed, achieving different, sometimes incomparable, guarantees. Here is a whirlwind tour through this landscape of strategies. One could sample an edge uniformly at random (using reservoir sampling), then count common neighbors of its endpoints [22]; or sample an edge uniformly and sample a vertex not incident to it [8]; or sample a subset of edges by independently picking each with a carefully adjusted probability p [28, 6]; or choose a random color for each vertex and collect all monochromatic edges [35]; or sample a subset of vertices at random and collect all edges incident to the sample [6]. One could collect two random subsets of vertices at different sampling rates and further sample edges between the two subsets [25]; or, as in a very recent algorithm, sample a subset of vertices at rate p and further sample edges incident to this sample at rate q , for well-chosen p and q [20]. Notice that in the just-mentioned algorithms, the sampling technique is not actively trying to “grow” a triangle around a sampled edge. Instead, elements of the graph (vertices or edges) are sampled indiscriminately (typically in one streaming pass) and the triangles seen inside the sample are counted in another pass. We shall call such sampling strategies *oblivious*.

Besides the above oblivious sampling strategies, another set of works used what we shall call *targeted sampling* strategies,² where information previously stored about the stream guides what gets sampled subsequently. Here is another quick tour through these. One could sample *wedges* (defined as length-2 paths) in the input graph, using a more sophisticated reservoir sampling approach [21]; or sample an edge uniformly and then sample a second edge that touches the first [36]; or sample a vertex with probability proportional to its squared degree, then sample two neighbors of that vertex [32]; or sample an edge uniformly and then sample a neighbor of the lower-degree endpoint of that edge [5].

There are also a handful of algorithms that add further twists on top of the sample-count-extrapolate framework. The algorithm of [7] combines the vertex coloring idea of [35] with ℓ_2 estimation sketches to obtain a solution that can handle *dynamic* graph streams, where each stream update may either insert or delete an edge. The algorithm of [13] combines multiple runs of [35] with a *heavy/light edge partitioning* technique: an edge is deemed “heavy” if it participates in “too many” triangles. A key observation is that the variance of an estimator constructed by oblivious sampling – which needs to be small in order to guarantee good results in small space – can be bounded better if no heavy edges are involved. On the other hand, triangles involving a heavy edge are easier to pick up (because there are many of them!) by randomly sampling vertices at a low rate. Thus, by carefully picking the threshold for heaviness, one can combine an algorithm that counts all-light triangles efficiently with one that counts heavy-edged triangles efficiently for a good overall space bound. In this way, [13] obtains a space bound of $O(\varepsilon^{-2.5} \log \delta^{-1} \text{polylog } n \cdot m/\sqrt{T})$, while [32] provides a tight dependence on ε (i.e., ε^{-2}) by achieving $\tilde{O}(m/\sqrt{T})$ space.

Separately, the aforementioned targeted sampling strategies of [32] and [5] provide 4-pass algorithms for estimating $T_3(G)$ using space $\tilde{O}(m^{3/2}/T)$. When T is large enough – specifically, $T = \Omega(m)$ – this space bound is better than the $\tilde{O}(m/\sqrt{T})$ bound obtained via heavy/light edge partitioning. By picking the better of the two algorithms, one obtains a space bound of $\tilde{O}(\min\{m^{3/2}/T, m/\sqrt{T}\})$. Both portions of this bound are tight, thanks to lower bounds of $\Omega(m^{3/2}/T)$ and $\Omega(m/\sqrt{T})$ that follow by reducing from the SET-DISJOINTNESS communication problem [13, 5].

The algorithm of [20] is optimal in a different sense: it runs in a single pass and $\tilde{O}((m/T)(\Delta_E + \sqrt{\Delta_V}))$ space, where Δ_E and Δ_V are as defined in Theorem 1. Each term in this bound is tight; reductions from DISJOINTNESS [5] and the BOOLEAN-HIDDEN-MATCHING problem [25] imply $\Omega(m\sqrt{\Delta_V}/T)$ lower bounds and a reduction from the INDEX communication problem implies an $\Omega(m\Delta_E/T)$ bound for 1-pass algorithms [6]. Note, however, that this result is incomparable to the multi-pass upper bounds noted above. It must be so: a lower bound of $\Omega(m^3/T^2)$ holds for 1-pass algorithms [5].

1.3 Other Related Work

This work is focused on streams that simply *list* the input hypergraph’s hyperedges. This is sometimes called the *insert-only* streaming model, in contrast to the *dynamic* or *turnstile* streaming model where the stream describes a sequence of (hyper)edge insertions or deletions. A small subset of works mentioned in Section 1.2 do provide results in a turnstile model. Besides these, there is the recent seminal work [23] that fully settles the complexity of triangle counting in turnstile streams for *constant-degree* graphs. This work also considers the very

² To be perfectly honest, the terms “targeted sampling” and “oblivious sampling” do not have precise technical definitions, but we hope the conceptual distinction is helpful to the reader as it was to us.

general problem of counting copies of an arbitrary fixed-size hypergraph motif M inside a large input hypergraph H , again in a turnstile setting. Because of the way their upper bound results depend on the structure of M , they cannot obtain sublinear-space solutions for counting k -simplices without a strong constant-degree assumption on the input H .

A handful of works on triangle counting consider adjacency list streams [8, 28, 32], where the input stream provides all edges incident to each vertex contiguously. This setting can somewhat simplify algorithm design, though the basic framework is still sample-count-extrapolate. We do not consider adjacency list streams in this work.

There are important related algorithms that predate the now-vast literature on streaming algorithms. In particular, [1] gives the current best run-time for exact triangle counting in the RAM model and [10] gives time-efficient algorithms for listing all triangles (and more general motifs). A version of the heavy/light partitioning idea appears in these early works.

More recently, a handful of works [15, 2, 16] have designed *sublinear-time* algorithms for approximately counting triangles and other motifs given query access to a large input graph. Triangle detection, listing, and counting have connections to other important problems in the area of fine-grained complexity [18, 43]. Triangle counting has also been studied in distributed, parallel, and high-performance computing models [40, 35, 39, 19].

2 Preliminaries

We now define our key terminology, set up notation, and establish some basic facts that we shall refer to in the algorithms and analyses to come.

► **Definition 3** (Hypergraph, degrees, neighborhoods). *A hypergraph is a pair $H = (V, E)$ where V is a nonempty finite set of vertices and $E \subseteq 2^V$ is a set of hyperedges. In certain contexts, this structure is instead called a set system over V . If $|e| = k \geq 1$ for all $e \in E$, then H is said to be k -uniform. For simplicity, we shall often shorten “ k -uniform hypergraph” to k -graph and “hyperedge” to “edge.”*

For each $S \subseteq V$ with $|S| < k$, the joint degree or codegree $\text{codeg}(S)$ of S is the number of edges that strictly extend S . The neighborhood $N(S)$ of S is the set of non- S vertices that share an edge with S . Formally,

$$\text{codeg}(S) := |\{e \in E : e \supseteq S\}|; \quad (1)$$

$$N(S) := \{v \in V : v \notin S \text{ and } \exists e \in E \text{ such that } e \supseteq S \cup \{v\}\}. \quad (2)$$

For a singleton set $S = \{u\}$, we define $\text{deg}(u) := \text{codeg}(\{u\})$ and $N(u) := N(\{u\})$. Further, given $S \subseteq V$, we define the S -relative degrees of vertices $x \in V \setminus S$ by

$$\text{deg}(x | S) := \text{codeg}(S \cup \{x\}). \quad (3)$$

Note that this is meaningful only when $|S| \leq k - 2$. Note, also, that $\text{deg}(x) = \text{deg}(x | \emptyset)$.

Throughout the paper, hypergraphs will be k -uniform unless qualified otherwise. We shall consistently use the notation $H = (V, E)$ for a generic k -graph and define $n := |V|$ and $m := |E|$. We shall assume that each vertex $v \in V$ has a unique ID, denoted $\text{ID}(v)$, which is an integer in the range $[n] := \{1, \dots, n\}$.

In general, there isn’t an equation relating $|N(S)|$ to $\text{codeg}(S)$. However, all k -graphs satisfy the following useful lemmas.

► **Lemma 4.** *For $S \subseteq V$ with $|S| < k$, $\text{codeg}(S) \leq |N(S)| \leq (k - |S|) \text{codeg}(S)$.*

Proof. Each edge that extends S adds ≥ 1 and $\leq k - |S|$ new neighbors to the sum. ◀

► **Lemma 5.** For $1 \leq r < k$, we have $\sum_{S \subseteq V: |S|=r} \text{codeg}(S) = \binom{k}{r} m = O(m)$.

Proof. The sum counts each edge exactly $\binom{k}{r}$ times. ◀

► **Definition 6 (Simplices).** A k -simplex is a k -graph on $k + 1$ vertices such that all possible k -sized edges are present. If $H = (V, E)$ is a k -graph and $X \subseteq V$ with $|X| = k + 1$, we say that H has a simplex at X if the induced subhypergraph $H[X] := (X, \{e \in E : e \subseteq X\})$ is a simplex. Abusing notation, we also use X to denote this simplex.

We use $\mathcal{T}_k(H)$ to denote the set of all k -simplices in H and $T_k(H) := |\mathcal{T}_k(H)|$ to denote the number of such simplices. We define $\Delta_E = \Delta_E(H)$ and $\Delta_V = \Delta_V(H)$ as follows:

$$\Delta_E := \max_{e \in E} |\{X \in \mathcal{T}_k(H) : X \text{ contains edge } e\}|; \quad (4)$$

$$\Delta_V := \max_{v \in V} |\{X \in \mathcal{T}_k(H) : X \text{ contains vertex } v\}|. \quad (5)$$

Drawing inspiration from the *link* operation for abstract and simplicial complexes in topology, we define a couple of operations that derive $(k - 1)$ -graphs from k -graphs.

► **Definition 7 (Neighborhood and shadow hypergraphs).** Let $k \geq 3$ and let $H = (V, E)$ be a k -graph. For each $u \in V$, the neighborhood hypergraph of H at u is the $(k - 1)$ -graph $H \downarrow u = (N(u), E_u)$, where

$$E_u := \{\{x_1, \dots, x_{k-1}\} : \{u, x_1, \dots, x_{k-1}\} \in E\} = \{e \setminus u : e \in E \text{ and } e \ni u\}.$$

The shadow hypergraph of H is the $(k - 1)$ -graph $H' = (V', E') = (\bigcup_{u \in V} V'_u, \bigcup_{u \in V} E'_u)$, where

$$V'_u := \{x^{(u)} : x \in V\} \text{ is a copy of } V \text{ “flavored” with } u, \text{ and}$$

$$E'_u := \{\{x_1^{(u)}, \dots, x_{k-1}^{(u)}\} : \{u, x_1, \dots, x_{k-1}\} \in E \text{ and } \text{ID}(u) \leq \text{ID}(x_i) \forall i \in [k - 1]\}.$$

Note that E'_u is subtly different from E_u in that it is induced by hyperedges incident on u in which u is the minimum-ID vertex. Observe that, as a result, $|E'| = |E|$.

We use neighborhood hypergraphs to analyze our first major algorithm, in Section 3. We use shadow hypergraphs in a crucial way to obtain our second major algorithm, in Section 4.2.

In the literature on triangle counting in graphs, the structure formed by two edges sharing a common vertex is called a *wedge*. It will be useful to define an analog of the notion for hypergraphs.

► **Definition 8 (Hyperwedge).** A k -hyperwedge is the hypergraph obtained by deleting one hyperedge from a k -simplex. Thus, if its vertex set is X , then $|X| = k + 1$ and it has k hyperedges with exactly one common vertex, which we call the apex of the hyperwedge. The only k -sized subset of X that is not present in the hyperwedge is called the base of the hyperwedge. Adding this base as a new hyperedge produces a k -simplex.

Fix a k -graph H and a k -simplex X in H . Observe that X contains $k + 1$ distinct hyperwedges; each such hyperwedge W corresponds to a $(k - 1)$ -simplex in the neighborhood hypergraph of H at the apex of W . On the other hand, X corresponds to exactly one $(k - 1)$ -simplex in the shadow hypergraph H' , namely the simplex at $\{u_1^{(z)}, \dots, u_k^{(z)}\}$, where z is the minimum-ID vertex in X and $\{u_1, \dots, u_k\} = X \setminus \{z\}$.

Our algorithm analyses often use the following standard boosting lemma from the theory of randomized algorithms. A random variable \tilde{Q} is said to be an (ε, δ) -estimate of a statistic Q if $\Pr[\tilde{Q} \in (1 \pm \varepsilon)Q] \geq 1 - \delta$.

► **Lemma 9** (Median-of-Means improvement). *If \hat{Q} is an unbiased estimator of some statistic, then one can obtain an (ε, δ) -estimate of that statistic by suitably combining*

$$K := \frac{C}{\varepsilon^2} \ln \frac{2}{\delta} \cdot \frac{\text{Var}[\hat{Q}]}{\mathbb{E}[\hat{Q}]^2}$$

independent samples of \hat{Q} , where C is a universal constant. ▮

3 An Optimal Algorithm Based on Targeted Sampling

In this section, we establish Subresult (1.1) of Theorem 1 by giving a formal description of the relevant algorithm (as Algorithm 1) followed by its analysis. This algorithm is based on “targeted sampling,” as outlined in Section 1.2. It runs in $\tilde{O}(m^{1+1/k}/T)$ space, which is optimal in view of Subresult (2.3) in Theorem 2. It is the method of choice for the “abundant” case, when T is large enough: specifically, in view of the guarantees of Algorithm 2 to follow (see Theorem 16), “large enough” should be defined as $T \geq m^{(k+1)/(k^2-k)}$.

The full version of the paper [9] additionally gives a more straightforward targeted-sampling algorithm, achieving a suboptimal space bound of $\tilde{O}(m^{2-1/k})$. Although suboptimal, that algorithm’s analysis contains novel ideas about the structure of hypergraphs that might be useful in subsequent research, such as the notion of “hyperabroricity.”

3.1 The Algorithm

The basic setup is as in the algorithm of [5] for counting odd cycles in graphs (and an analogous algorithm of [32]). We use one pass to pick an edge $e \in E$ uniformly at random and, in subsequent passes, use further sampling to estimate the number of suitable apex vertices z at which there is a hyperwedge with base e , which would complete a simplex together with e . In order to control the variance of the resulting unbiased estimator, we want the vertices of a detected simplex $e \cup \{z\}$ to respect a certain ordering “by degree.”

We let the sampled edge e determine the ordering, for which we use *relative* degrees, relative to certain subsets of e . Moreover, this ordering – which we call the e -relative ordering – is only partial, as we never need to compare two vertices that both lie outside e . Let $c_i(e)$ be the i th vertex of e according to this ordering that we shall soon define. In our algorithm, we shall look for a suitable apex z only in the neighborhood $N(\{c_1(e), \dots, c_{k-1}(e)\})$. Furthermore, we shall require that z have a larger degree than $c_1(e)$, a larger $\{c_1(e)\}$ -relative degree than $c_2(e)$, a larger $\{c_1(e), c_2(e)\}$ -relative degree than $c_3(e)$, and so on. As we shall see, the analysis of the resulting algorithm uses the recursive structure of k -graphs and k -simplices through the notion of neighborhood hypergraphs (Definition 7).

These next two, somewhat elaborate, definitions formalize the above ideas.

► **Definition 10** (e -relative ordering). *Fix a hyperedge $e \in E$. The e -relative ordering, \prec_e , is a partial order on V defined as follows. Set $S_0(e) = \emptyset$ and define the following, iteratively, for i running from 1 to k .*

Let $c_i(e)$ be the vertex in $e \setminus S_{i-1}(e)$ that minimizes $\deg(c_i(e) \mid S_{i-1}(e))$, with ties resolved in favor of the vertex with smallest ID. Set $S_i(e) := S_{i-1}(e) \cup \{c_i(e)\} = \{c_1(e), \dots, c_i(e)\}$.

Then declare $c_1(e) \prec_e \cdots \prec_e c_k(e)$. Further, for each $z \in V \setminus e$, declare $c_k(e) \prec_e z$ if the following two things hold: (a) for $1 \leq i \leq k-1$, we have $\deg(c_i(e) \mid S_{i-1}(e)) \leq \deg(z \mid S_{i-1}(e))$, and (b) we also have $\deg(c_k(e) \mid S_{k-2}(e)) \leq \deg(z \mid S_{k-2}(e))$. Ties in degree comparisons are resolved by declaring the vertex with smaller ID to be smaller.³

► **Definition 11** (Simplex labeling). Suppose that H has a simplex at X , where $X \subseteq V$. Number the vertices in X as u_1, \dots, u_{k+1} iteratively, as follows. For i from 1 to $k-1$, let u_i be the vertex in X that minimizes $\deg(u_i \mid \{u_1, \dots, u_{i-1}\})$. Let u_k be the vertex among $\{u_k, u_{k+1}\}$ that minimizes $\deg(u_k \mid \{u_1, \dots, u_{k-2}\})$. This automatically determines u_{k+1} . Based on this, define the label of the simplex at X to be $(e(X), z(X))$, where $e(X) := \{u_1, \dots, u_k\}$ and $z(X) := u_{k+1}$.

Notice that if a simplex is labeled (e, z) , then the e -relative ordering satisfies $\max_{\prec_e}(e) \prec_e z$. Conversely, if there is a simplex at X and an edge e of this simplex such that $X = e \cup \{z\}$ and $\max_{\prec_e}(e) \prec_e z$, then the label of this simplex is (e, z) .

For each $e \in E$, let τ_e denote the number of simplices that have a label of the form (e, \cdot) . Thanks to the uniqueness of labels, we immediately have

$$\sum_{e \in E} \tau_e = T_k(H). \quad (6)$$

Our algorithm is described in Algorithm 1. Our analysis will make use of the following two combinatorial lemmas at key moments. The proofs of these lemmas can be found in the full paper [9].

► **Lemma 12.** For every edge $e \in E$, we have $\tau_e \leq km^{1/k}$.

► **Lemma 13.** If H is k -uniform and $k \geq 3$, then $\sum_{e \in E} \text{codeg}(S_{k-1}(e)) = O(m^{1+1/k})$.

3.2 Analysis of the Algorithm

We begin by proving that the estimator Y computed by Algorithm 1 is unbiased. Let \mathcal{E}_e denote the event that the edge sampled in pass 1 is e . For each $j \in [R]$, the vertex x_j is picked uniformly at random from $N(S_{k-1}(e))$ and exactly τ_e choices cause Z_j to be set to the nonzero value $\text{codeg}(S_{k-1}(e))$. Therefore,

$$\mathbb{E}[Z_j \mid \mathcal{E}_e] = \frac{\tau_e}{|N(S_{k-1}(e))|} \cdot \text{codeg}(S_{k-1}(e)) = \tau_e \quad (7)$$

because $\text{codeg}(S_{k-1}(e)) = |N(S_{k-1}(e))|$, by Lemma 4. By the law of total expectation and Equation (6),

$$\mathbb{E}[Y] = \frac{m}{R} \sum_{j=1}^R \sum_{e \in E} \frac{1}{m} \mathbb{E}[Z_j \mid \mathcal{E}_e] = \frac{1}{R} \sum_{j=1}^R \sum_{e \in E} \tau_e = T_k(H). \quad (8)$$

Next, we bound the variance of the estimator. Proceeding along similar lines, we have $\mathbb{E}[Z_j^2 \mid \mathcal{E}_e] = \text{codeg}(S_{k-1}(e)) \cdot \tau_e$ and $\mathbb{E}[Z_{j_1} Z_{j_2} \mid \mathcal{E}_e] = \tau_e^2$ for all $j_1 \neq j_2$, because Z_{j_1} and Z_{j_2} are independent conditioned on \mathcal{E}_e . So,

³ This is how ties are resolved from now on, even when not mentioned.

■ **Algorithm 1** Counting k -simplices in k -uniform hypergraph streams: the “abundant” case.

```

1: procedure  $k$ -SIMPLEX-COUNT-ABUNDANT( $\sigma$  : stream of edges of  $k$ -graph  $H = (V, E)$ )
2:   pass 1: ▷  $O(1)$  space
3:     pick edge  $e = \{u_1, u_2, \dots, u_k\} \in E$  u.a.r. using reservoir sampling
4:   pass 2: ▷  $O(2^k) = O(1)$  space
5:     compute  $\text{codeg}(S)$  for all non-trivial  $S \subset e$ 
6:   pass 3: ▷  $O(R)$  space
7:     re-arrange (if needed)  $u_1, \dots, u_k$  so that  $u_i = c_i(e)$  for  $i \in [k]$ , using co-degrees
       from pass 2
8:      $R \leftarrow \lceil \text{codeg}(S_{k-1}(e)) \cdot m^{-1/k} \rceil$  ▷ note that  $\text{codeg}(S_{k-1}(e)) = |N(S_{k-1}(e))|$ 
9:     for  $j \leftarrow 1$  to  $R$ , in parallel, do
10:       $Z_j \leftarrow 0$ 
11:      pick vertex  $x_j \in N(S_{k-1}(e))$  u.a.r. using reservoir sampling from relevant
       substream of  $\sigma$ 
12:   pass 4: ▷  $O(kR) = O(R)$  space
13:     compute the relative degrees  $\text{deg}(x_j \mid S_{i-1}(e))$  for all  $i \in [k-1]$  and  $j \in [R]$ 
14:     for  $j \leftarrow 1$  to  $R$ , in parallel, do
15:       if  $e \cup \{x_j\}$  determines a simplex with label  $(e, x_j)$  then
16:          $Z_j \leftarrow \text{codeg}(S_{k-1}(e))$ 
17:   return  $Y \leftarrow (m/R) \sum_{j=1}^R Z_j$  ▷ Average out the trials and scale

```

$$\begin{aligned} \mathbb{E}[Y^2 \mid \mathcal{E}_e] &= \mathbb{E} \left[\left(\frac{m}{R} \sum_{j=1}^R Z_j \right)^2 \mid \mathcal{E}_e \right] = \frac{m^2}{R^2} \sum_{j=1}^R \mathbb{E}[Z_j^2 \mid \mathcal{E}_e] + \frac{m^2}{R^2} \sum_{j_1 \neq j_2} \mathbb{E}[Z_{j_1} Z_{j_2} \mid \mathcal{E}_e] \\ &\leq \frac{m^2 \cdot \text{codeg}(S_{k-1}(e)) \cdot \tau_e}{R} + \frac{m^2 R(R-1) \tau_e^2}{R^2} \leq m^{2+1/k} \tau_e + m^2 \tau_e^2, \end{aligned}$$

since $\text{codeg}(S_{k-1}(e)) \leq m^{1/k} R$ by the definition of R . Using $\sum_{e \in E} \tau_e = T_k(H)$, we get

$$\begin{aligned} \text{Var}[Y] &\leq \mathbb{E}[Y^2] = \sum_{e \in E} \frac{1}{m} \mathbb{E}[Y^2 \mid \mathcal{E}_e] \leq m^{1+1/k} \sum_{e \in E} \tau_e + m \sum_{e \in E} \tau_e^2 \\ &\leq m^{1+1/k} T_k(H) + m \cdot (\max_{e \in E} \tau_e) \cdot \sum_{e \in E} \tau_e = O\left(m^{1+1/k} T_k(H)\right), \end{aligned}$$

where we invoked Lemma 12 and used Equation (6) twice.

Having bounded the variance, we may apply the median-of-means technique (Lemma 9) to the basic estimator of Algorithm 1 to obtain a final algorithm that provides an (ε, δ) -estimate. As noted in the comments within Algorithm 1, the basic estimator uses $O(R)$ space, where R is a random variable. Therefore, the final algorithm, which still uses four passes, has expected space complexity $\tilde{O}(\mathbb{E}[R] \cdot \text{Var}[Y]/\mathbb{E}[Y]^2) = \mathbb{E}[R] \cdot \tilde{O}(m^{1+1/k}/T_k(H)) = \mathbb{E}[R] \cdot \tilde{O}(m^{1+1/k}/T)$, thanks to the promise that $T_k(H) \geq T$.

Finally, we need to bound R in expectation. We find that

$$\mathbb{E}[R] = \frac{1}{m} \sum_{e \in E} \mathbb{E}[R \mid \mathcal{E}_e] = \frac{1}{m} \sum_{e \in E} \left\lceil \frac{\text{codeg}(S_{k-1}(e))}{m^{1/k}} \right\rceil \leq 1 + m^{-1-1/k} \sum_{e \in E} \text{codeg}(S_{k-1}(e)).$$

Invoking Lemma 13, we conclude that $\mathbb{E}[R] = O(1)$, giving our first main algorithmic result.

► **Theorem 14.** *There is a 4-pass algorithm that reads a stream of m hyperedges specifying a k -uniform hypergraph H and produces an (ε, δ) -estimate of $T_k(H)$; under the promise that $T_k(H) \geq T$, the algorithm runs in expected space $O(\varepsilon^{-2} \log \delta^{-1} \log n \cdot m^{1+1/k}/T)$. ◻*

We conclude with two remarks about this algorithm. First, the space bound can easily be made worst-case, by applying the following modification to the median-of-means boosting process. Notice that in each instance of the basic estimator, at the end of pass 2, we know the value of R and thus the actual space that this instance would use. For each batch of estimators for which we compute a mean, we determine whether this batch would use more than A times its expected space bound, for some large constant A . If so, we abort this batch and don't use it in the median computation. By Markov's inequality, the probability that we abort a batch is at most $1/A$, which does not affect the rest of the standard analysis of the quality of the final median-of-means estimator.

Second, while we have not dwelt on *time* complexity in our exposition, it is not hard to convince oneself that the algorithm processes each incoming edge in $\tilde{O}(R)$ time, which is $\tilde{O}(1)$ in expectation.

4 Algorithms Based on Oblivious Sampling

We now turn to our second family of algorithms, which use “oblivious sampling,” as outlined in Section 1.2. To recap, whereas the algorithms in the previous section used an initially-sampled edge to guide their subsequent sampling, an oblivious sampling strategy decides whether or not to store an edge based only coin tosses specific to that edge and perhaps its constituent vertices (and not on any edges in store). The main result of this section is an $\tilde{O}(m/T^{2/(k+1)})$ -space algorithm, given as Algorithm 2, which is the method of choice for the “meager” case, when T is not too large (specifically, $T < m^{(k+1)/(k^2-k)}$). Simpler versions of our ideas give algorithms with worse space guarantees: namely, $\tilde{O}(m/T^{1/k})$ and $\tilde{O}(m/T^{2/(k+2)})$. The full version of the paper [9] gives the details.

The full paper additionally describes a 1-pass algorithm – also based on oblivious sampling – whose space guarantee depends on additional structural parameters of the input hypergraph H , proving Subresult (1.3) of Theorem 1.

4.1 A Unifying Framework for Oblivious Sampling Strategies

We find it useful to set up a framework for analyzing algorithms based on oblivious sampling and view our eventual algorithm as an instantiation of the framework. Further, this framework unifies a number of approaches seen in previous work on triangle counting [32, 20, 25, 6].

Suppose that we are trying to estimate the number, $T(H)$, of copies of a target substructure (motif) Ξ in a streamed hypergraph H . Suppose we do this by collecting certain edges into a random sample Q , using some random process parameterized by a quantity p . The sample Q determines which of the $T(H)$ copies of Ξ get “detected” by the rest of the logic of the algorithm. Let $\Xi_1, \dots, \Xi_{T(H)}$ be the copies of Ξ in H (under some arbitrary enumeration scheme) and let Λ_i be the indicator random variable for the event that Ξ_i is detected.

In our framework, we will want to identify parameters α, β , and γ such that the following conditions hold for all $i, j \in [T(H)]$ with $i \neq j$ and all $e \in E$:

$$\Pr[\Lambda_i = 1] = p^\alpha; \quad \Pr[\Lambda_i = \Lambda_j = 1] \leq p^\beta; \quad \Pr[e \in Q] = p^\gamma. \quad (9)$$

In instantiations of the framework, we will of course have $\beta > \alpha$ and further, we will have $\beta < 2\alpha$, which is natural because for certain pairs i, j , the variables Λ_i and Λ_j will be positively correlated. We shall further require that our detection process satisfy the *separation property*, where $\Xi \cap \Xi'$ denotes the set of *edges* common to Ξ and Ξ' and the notation “ $X \perp Y$ ” means that random variables X and Y are independent.

SEPARATION PROPERTY: If $\Xi_i \cap \Xi_j = \emptyset$, then $\Lambda_i \perp \Lambda_j$. (10)

The logic of the algorithm will count the number of detected copies of Ξ in an accumulator $A := \sum_{i=1}^{T(H)} \Lambda_i$. Defining the estimator $\tilde{T} := A/p^\alpha$, we see that $\mathbb{E}[\tilde{T}] = \sum_{i=1}^{T(H)} p^{-\alpha} \mathbb{E}[\Lambda_i] = T(H)$, which makes it unbiased. To establish concentration, we estimate

$$\begin{aligned} \text{Var}[A] &= \text{Var} \sum_{i=1}^{T(H)} \Lambda_i = \sum_{i=1}^{T(H)} \text{Var}[\Lambda_i] + \sum_{i \neq j} \text{Cov}[\Lambda_i, \Lambda_j] \leq \sum_{i=1}^{T(H)} \mathbb{E}[\Lambda_i^2] + \sum_{i \neq j} \text{Cov}[\Lambda_i, \Lambda_j] \\ &\leq T(H)p^\alpha + \sum_{\Lambda_i \not\perp \Lambda_j} \mathbb{E}[\Lambda_i \Lambda_j] \leq T(H)p^\alpha + \sum_{\Lambda_i \not\perp \Lambda_j} p^\beta = T(H)p^\alpha + p^\beta \cdot \underbrace{\sum_{e \in E} \sum_{\Xi_i \cap \Xi_j = \{e\}} 1}_{s_{\text{cor}}}, \end{aligned} \quad (11)$$

where the penultimate step uses (9) and the last step holds because of the separation property and the reasonable structural assumption (true of simplices) that two distinct copies of the motif Ξ can intersect in at most one edge.

The term s_{cor} captures the amount of correlation in the sampling process. Let us now specialize the discussion to the problem at hand, where the motif Ξ is a k -simplex and so, $T(H) = T_k(H)$. Let M_e be the number of simplices that share hyperedge e (note that this is subtly different from τ_e in Section 3). Recalling the definition of Δ_E in Equation (4), we see that $\max_{e \in E} M_e = \Delta_E$. We can therefore upper bound s_{cor} as follows:

$$s_{\text{cor}} \leq \sum_{e \in E} \binom{M_e}{2} \leq \sum_{e \in E} \frac{M_e^2}{2} \leq \frac{\Delta_E}{2} \sum_{e \in E} M_e = \frac{(k+1)\Delta_E T_k(H)}{2}. \quad (12)$$

Plugging this bound into (11) and applying Chebyshev’s inequality gives the following “error probability” bound, where $B > 0$ will be chosen later.

$$P_{\text{err}}^{(B)}[\tilde{T}] := \Pr \left[|\tilde{T} - T(H)| \geq \varepsilon B \right] \leq \frac{\text{Var}[A]}{p^{2\alpha} \varepsilon^2 B^2} \leq \frac{T_k(H)}{\varepsilon^2 p^\alpha B^2} + \frac{(k+1)\Delta_E T_k(H)}{2\varepsilon^2 p^{2\alpha-\beta} B^2}. \quad (13)$$

Heavy/Light Edge Partitioning via an Oracle. Using the above estimator \tilde{T} directly on the input hypergraph H does not give a good bound in general, because the quantity Δ_E appearing in Equation (13) could be as large as $T_k(H)$, making the upper bound on P_{err} useless. One remedy is to apply the technique so far to a subgraph of H where Δ_E is *guaranteed* to be a fractional power of $T_k(H)$ and deal separately with simplices that don’t get counted in such a subgraph. Such a subgraph can be obtained by taking only the “light” edges in H , where an edge e is considered to be light if $M_e = O(T_k(H)^\theta)$, for some parameter $\gamma < 1$ that we will soon choose. We remark that this kind of heavy/light partitioning is a standard technique in this area: it occurs in the triangle counting algorithms of [12, 32] and in fact in the much earlier triangle listing algorithm of [10].

More precisely, we create an *oracle* to label each edge as either HEAVY or LIGHT. Thus:

$$T_k(H) = T_k^L(H) + T_k^H(H), \quad (14)$$

32:12 Counting Simplices in Hypergraph Streams

where $T_k^L(H)$ is the number of simplices containing only light edges, and $T_k^H(H)$ is the number of simplices containing at least one heavy edge. The key insight in the partitioning technique is that we can estimate the two terms in Equation (14) separately, using the estimator \tilde{T} described above for $T_k^L(H)$ and a different estimator for $T_k^H(H)$ that we shall soon describe. In what follows, we allow an additive error of $\pm \varepsilon T_k(H)$ in the estimate of each term, leading to a multiplicative error of $1 \pm 2\varepsilon$ in the estimation of $T_k(H)$.

The oracle is implemented by running the following one-pass randomized streaming algorithm implementing a function $\text{orcl}: E \rightarrow \{\text{HEAVY}, \text{LIGHT}\}$ to label the edges of H . Form a random subset $Z \subseteq V$ by picking each vertex $v \in V$, independently, with probability

$$q := \xi \varepsilon^{-2} \ln n \cdot T^{-\theta}, \quad (15)$$

where T is the promised lower bound on $T_k(H)$ given to the algorithm, and $\xi \geq 12k(k+1)$ is a constant. Then, in one pass over the input stream, collect all hyperedges e containing at least one vertex from Z . Let S denote the random sample of edges thus collected. Note that this is distinct from (and independent of) the sample Q collected by the oblivious sampling scheme discussed above. Now, for each hyperedge $e = \{u_1, \dots, u_k\} \in E$, let $\widetilde{M}_e := |\{z \in Z : e \cup \{z\} \in \mathcal{T}_k(H)\}|$ be the number of simplices that e completes with respect to Z . Then we define:

$$\text{orcl}(e) = \begin{cases} \text{HEAVY}, & \text{if } \widetilde{M}_e < qT^\theta, \\ \text{LIGHT}, & \text{otherwise.} \end{cases} \quad (16)$$

The next lemma (whose proof, via Chernoff bounds, we omit) says that the oracle's predictions correspond, with high probability, to our intuitive definition of heavy/light edges, up to a factor of 2.

► **Lemma 15.** *W.h.p., the oracle given by (16) is “correct,” meaning that it satisfies*

$$\forall e \in E : \text{orcl}(e) = \text{LIGHT} \Rightarrow M_e < 2T^\theta \quad \wedge \quad \text{orcl}(e) = \text{HEAVY} \Rightarrow M_e > \frac{1}{2}T^\theta.$$

The space required to implement the oracle is that required to store the sample S . Since

$$\mathbb{E}[|S|] = O\left(\sum_{v \in V} q \deg(v)\right) = O(qm), \quad (17)$$

the expected space is $O(qm \log n) = O(\varepsilon^{-2} \log^2 n \cdot T^{-\theta} m) = \tilde{O}(m/T^\theta)$, by Equation (15).

Counting the Light and Heavy Simplices. We can now describe our overall algorithm. We use two streaming passes. In the first pass, we collect the samples S and Q . At the end of the pass, we have the information necessary to prepare the heavy/light labeling oracle and using it, we remove all heavy edges from Q . In the second pass, for each edge e encountered, we use the oracle to classify it. If e is light, we feed it into the detection logic for the oblivious sampling scheme (which uses Q). If e is heavy, we use S to estimate M_e , the number of simplices containing e ; this requires a little care, as we explain below.

In analyzing the algorithm, we start by assuming that the oracle is correct (which happens w.h.p., by Lemma 15). Let \tilde{T}^L be the estimator for $T_k^L(H)$ produced by the oblivious sampling scheme. We analyze its error probability using Equation (13), noting that $\Delta_E \leq 2T^\theta$ in the light-edges subgraph and choosing $B = T_k(H)$. This leads to

$$\begin{aligned}
P_{\text{err}}[\tilde{T}^L] &:= \Pr \left[|\tilde{T}^L - T_k^L(H)| \geq \varepsilon T_k(H) \right] \\
&\leq \frac{T_k^L(H)}{\varepsilon^2 p^\alpha T_k(H)^2} + \frac{(k+1) \cdot 2T^\theta \cdot T_k^L(H)}{2\varepsilon^2 p^{2\alpha-\beta} T_k(H)^2} \leq \frac{1}{\varepsilon^2 p^\alpha T} + \frac{k+1}{\varepsilon^2 p^{2\alpha-\beta} T^{1-\theta}}, \quad (18)
\end{aligned}$$

where we used the inequalities $T_k^L(H) \leq T_k(H)$ and $T_k(H) \geq T$. This bound on the error probability will inform our choices of parameters p, α, β , and θ , below.

Finally, we estimate $T_k^H(H)$. Define a simplex in $\mathcal{T}_k(H)$ to be *i-heavy* if it has exactly i heavy edges. For each edge e and $1 \leq i \leq k+1$, let M_e^i be the number of i -heavy simplices containing e . Notice that

$$\sum_{e \text{ heavy}} M_e^i = i \cdot |\{\Xi \in \mathcal{T}_k(H) : \Xi \text{ is } i\text{-heavy}\}|, \quad \text{whence } T_k^H(H) = \sum_{e \text{ heavy}} \sum_{i=1}^{k+1} \frac{M_e^i}{i}. \quad (19)$$

During our second streaming pass, we compute unbiased estimators for each M_e^i as follows. Recall that S is the set of all edges incident to some vertex in the set Z of sampled vertices. Define \tilde{M}_e^i to be the number of vertices in Z that combine with e to form an i -heavy simplex; we consult the oracle to test whether a simplex in question is indeed i -heavy. Notice that $\tilde{M}_e^i \sim \text{Bin}(M_e^i, q)$. Therefore, by Equation (19),

$$\tilde{T}^H := \frac{1}{q} \sum_{e \text{ heavy}} \sum_{i=1}^{k+1} \frac{\tilde{M}_e^i}{i} \quad (20)$$

is an unbiased estimator for $T_k^H(H)$. To establish concentration, consider the inner sum corresponding to any particular edge e . The sum can be rewritten as a sum of M_e mutually independent indicator variables scaled by factors in $\{1, 1/2, \dots, 1/(k+1)\}$. Since the oracle is correct, the heaviness of e implies that $M_e \geq \frac{1}{2}T^\theta$ and then a Chernoff bound gives

$$\Pr \left[\sum_{i=1}^{k+1} \frac{\tilde{M}_e^i}{i} \notin (1 \pm \varepsilon) q \sum_{i=1}^{k+1} \frac{M_e^i}{i} \right] \leq 2 \exp \left(-\frac{1}{3} \varepsilon^2 q \cdot \frac{1}{2} T^\theta \cdot \frac{1}{k+1} \right) \leq n^{-2k},$$

where the final step uses $\xi \geq 12k(k+1)$. By a union bound over the $\leq n^k$ heavy edges,

$$\begin{aligned}
P_{\text{err}}[\tilde{T}^H] &:= \Pr \left[|\tilde{T}^H - T_k^H(H)| \geq \varepsilon T_k(H) \right] \leq \Pr \left[\tilde{T}^H \notin (1 \pm \varepsilon) T_k^H(H) \right] \\
&\leq \Pr \left[\bigvee_{e \text{ heavy}} \left\{ \sum_{i=1}^{k+1} \frac{\tilde{M}_e^i}{i} \notin (1 \pm \varepsilon) q \sum_{i=1}^{k+1} \frac{M_e^i}{i} \right\} \right] \leq n^{-O(1)}. \quad (21)
\end{aligned}$$

Determining the Parameters and Establishing a Space Bound. We have now described the overall logic of a generic algorithm in our framework. To instantiate it, we must plug in a specific oblivious sampling scheme, whose logic will determine the parameters α, β , and γ as given by Equation (9). Now we choose p to ensure that $P_{\text{err}}[\tilde{T}^L]$ is less than a small constant. Thanks to Equation (18), it suffices to ensure that

$$\frac{1}{\varepsilon^2 p^\alpha T} + \frac{k+1}{\varepsilon^2 p^{2\alpha-\beta} T^{1-\theta}} = o(1).$$

32:14 Counting Simplices in Hypergraph Streams

Setting $p = (\log n / (\varepsilon^2 T))^{1/\alpha}$ reduces the first term to $1/\log n$. To make the second term small as well, we set the threshold parameter θ used by the heavy/light partitioning oracle to $\theta := \beta/\alpha - 1$; as we remarked after Equation (9), we will have $\alpha < \beta < 2\alpha$, which implies $0 < \theta < 1$. The second term now reduces to

$$\frac{(k+1)p^{\beta-\alpha}T^\theta}{\log n} = \frac{k+1}{(\log n)^{1-\theta}\varepsilon^{2\theta}} = o(1),$$

for n sufficiently large. We therefore obtain $P_{\text{err}}[\tilde{T}^L] = o(1)$ and, putting this together with Equation (21) and Lemma 15, we conclude that the overall algorithm computes a $(1 \pm 2\varepsilon)$ -approximation to $T_k(H)$ with probability at least $1 - o(1)$.

With these parameters now set, the space complexity analysis is straightforward. The space usage is dominating by the storing of the samples S and Q . By Equation (17) and Equation (15), we have $\mathbb{E}[|S|] = \tilde{O}(m/T^\theta)$ and by Equation (9), we have

$$\mathbb{E}[|Q|] = p^\gamma m = \left(\frac{\log n}{\varepsilon^2}\right)^{\gamma/\alpha} \frac{m}{T^{\gamma/\alpha}}.$$

It is natural that the probability of detecting a particular simplex is at most the probability of storing one particular edge, so $\gamma \leq \alpha$. Therefore, $\mathbb{E}[|Q|] \leq \varepsilon^{-2} \log n \cdot m/T^{\gamma/\alpha}$. Altogether, since storing a single edge uses $O(\log n)$ bits, the total space usage is $\tilde{O}(m/T^\lambda)$, where

$$\lambda = \min \left\{ \frac{\gamma}{\alpha}, \theta \right\} = \min \left\{ \frac{\gamma}{\alpha}, \frac{\beta}{\alpha} - 1 \right\}. \quad (22)$$

This completes the description of our framework for designing and analyzing simplex counting algorithms based on oblivious sampling.

4.2 Instantiation: Oblivious Sampling Using the Shadow Hypergraph

It is time to instantiate our framework with a specific sampling and simplex-detection strategy. The simplest instantiation samples edges at rate p and detects light simplices composed of sampled edges, leading to a $\tilde{O}(m/T^{1/k})$ space bound. We get a better bound of $\tilde{O}(m/T^{2/(k+2)})$ algorithm by adapting the vertex-coloring-based graph algorithm of [35]: a key insight is that we color $(k-1)$ -sized *subsets* of V and then pick fully monochromatic edges in the sample. Coloring single vertices would have run into the technical problem that the separation property would not hold. Our full paper [9] describes these ideas in detail.

Our best algorithm combines ideas from these initial approaches, eventually achieving a space complexity of $\tilde{O}(m/T^{2/(k+1)})$, which is our best bound of the form $\tilde{O}(m/T^\lambda)$. It is therefore our method of choice for the “meager” case. Comparing with Theorem 14, we see that this bound wins when $T < m^{(k+1)/(k^2-k)}$. We continue to use the coloring idea with two added twists: we color appropriately-sized *subsets* of vertices, and we do so with vertices of the *shadow hypergraph* $H' = (V', E')$, defined in Definition 7.

To be more specific, we uniformly and independently color $(k-2)$ -sized subsets of vertices V' of H' using N colors. Since each vertex in V' is essentially an ordered pair of vertices in V , we need a coloring function that maps $\binom{V \times V}{k-2}$ to $[N]$. Recall that $H' = (V', E')$ is a $(k-1)$ -uniform hypergraph. During the first pass, when an edge $e = \{u_1, \dots, u_k\}$ arrives in the stream, it implies the arrival of an edge $e' = E'$ which we store in Q iff all $(k-2)$ -sized subsets within e' receive the same color. We call such an edge e' *monochromatic*.

We detect simplices using this sample Q as follows. As observed after Definition 8, each k -simplex in H corresponds to exactly one $(k-1)$ -simplex in H' ; we try to detect this “shadow simplex.” In greater detail, let Ξ be a particular simplex in H . Let z be its minimum-ID vertex, let e_z be the edge of Ξ not incident to z and let W_z be the hyperwedge

of Ξ with apex z . Note that the edges of W_z correspond to edges in H' that lie in E'_z (see Definition 7) and, since W_z is a hyperwedge, these edges form a $(k-1)$ -simplex in the z -flavored component of H' . In the second pass, when e_z arrives in the stream, we detect Ξ iff this entire $(k-1)$ -simplex has been stored in Q , i.e., iff all its edges are monochromatic. Algorithmically, when we see an edge e in the second pass, we go over all vertices $z \in V$, detecting a simplex whenever this e plays the role of e_z for a simplex on the vertices in $e \cup \{z\}$ in the manner just described.

The remaining details are as described in Section 4.1: we apply the heavy/light edge partitioning technique on top of the above sampling scheme and proceed as in our framework. We formalize the overall algorithm, with full details, as Algorithm 2. Note that the coloring function need not be fully random and can be chosen from a d -wise independent hash family for an appropriate constant d . The update logic for the accumulator A_L (Section 4.2) implements the detection method just described. The update logic for the accumulator A_H (Section 4.2) implements the estimator in Equation (20). Note that the heavy-simplex estimator does not use the shadow hypergraph.

■ **Algorithm 2** Simplex counting in the “meager” case using $\tilde{O}(m/T^{2/(k+1)})$ bits of space.

```

1: procedure  $k$ -SIMPLEX-COUNT-MEAGER( $\sigma$  : stream of edges of  $k$ -graph  $H = (V, E)$ )
2:    $d \leftarrow 2\binom{k}{k-2}$ ;  $r \leftarrow \binom{n^2}{k-2}$ ;  $\alpha \leftarrow \binom{k}{k-2} - 1$ ;  $p \leftarrow (\varepsilon^{-2}T^{-1} \log n)^{1/\alpha}$ ;  $N \leftarrow \lceil 1/p \rceil$ 
3:    $\theta \leftarrow (2\binom{k}{2} - k) / (\binom{k}{2} - 1)$ ;  $q \leftarrow \xi \varepsilon^{-2} \log n \cdot T^{-\theta}$ 
4:    $\triangleright \xi$  is a suitable constant; see Equation (15)
5:    $Z \leftarrow$  sample of  $V$ , picking each vertex independently with probability  $q$ 
6:    $\mathcal{F} \subseteq [N]^{[r]} \leftarrow$   $d$ -wise independent family of hash functions of the form  $h: [r] \rightarrow [N]$ 
7:   select a function  $\text{col}(\cdot)$  uniformly at random from  $\mathcal{F}$  to color the elements of  $\binom{V'}{k-2}$ 
8:   pass 1:
9:      $(Q, S) \leftarrow (\emptyset, \emptyset)$ 
10:    for each hyperedge  $e = \{u_1, \dots, u_k\}$  in the stream do
11:      let  $i \in [k]$  be such that  $u_i$  is the vertex in  $e$  with the smallest ID
12:      if all  $(k-2)$ -sized subsets of  $e \setminus \{u_i\} \in E_{u_i}$  are colored the same then
13:         $Q \leftarrow Q \cup \left\{ \{u_1^{(u_i)}, \dots, u_{i-1}^{(u_i)}, u_{i+1}^{(u_i)}, \dots, u_k^{(u_i)}\} \right\}$ 
14:      if  $e \cap Z \neq \emptyset$  then
15:         $S \leftarrow S \cup \{e\}$ 
16:    create heavy/light oracle based on  $S$ 
17:   pass 2:
18:      $(A_L, A_H) \leftarrow (0, 0)$ 
19:     let  $Q^L$  be the set of hyperedges in  $Q$  corresponding to light hyperedges in  $E$ 
20:     for each hyperedge  $e = \{u_1, \dots, u_k\}$  in the stream do
21:       if  $\text{orcl}(e) = \text{LIGHT}$  then
22:          $A_L \leftarrow A_L + \left| \left\{ z \in V : \bigwedge_{i=1}^k \left\{ \{u_1^{(z)}, \dots, u_{i-1}^{(z)}, u_{(i+1)}^{(z)}, \dots, u_k^{(z)}\} \subseteq Q^L \right\} \right\} \right|$ 
23:       else
24:         Let  $\hat{M}_e^i := |\{z \in Z : e \cup \{z\} \text{ a simplex with } i \text{ heavy edges}\}|$ 
25:          $A_H \leftarrow A_H + \sum_{i=1}^{k+1} \hat{M}_e^i / i$ 
26:   return  $A_L/p^\alpha + A_H/q$ 

```

Analysis. We proceed in the now-established manner within our framework: we need to work out the parameters α , β , and γ .

Consider a particular k -simplex Ξ in H and let the $(k-1)$ -simplex Ξ' be its “shadow simplex” in H' , as in the above discussion. Then, if Ξ is light, it is detected at most once, when the hyperedge opposite to its lowest-ID vertex arrives in the stream. For this detection to actually happen, all edges of Ξ' must be monochromatic, i.e., all $\binom{k}{k-2}$ possible $(k-2)$ -sized subsets of vertices in Ξ' must receive the same color. Let $p = 1/N$. The probability of this event is p^α , where $\alpha = \binom{k}{k-2} - 1 = \binom{k}{2} - 1$.

Next, we observe that for two distinct simplices Ξ_1 and Ξ_2 in H , the events of their detection have nonzero correlation iff they have an edge in common. Thus, the separation property (10) holds. Moreover, when $\Xi_1 \cap \Xi_2 = \{e\}$, the correlation is in fact nonzero only if both Ξ_1 and Ξ_2 have the same minimum-ID vertex and that vertex lies in e . When this happens, the simultaneous detection event occurs iff all $(k-2)$ -sized subsets of shadow vertices arising from vertices of Ξ_1 and Ξ_2 receive the same color. Counting the number of such subsets shows that the probability of simultaneous detection is p^β , for

$$\beta = 2 \binom{k}{k-2} - (k-1) - 1 = 2 \binom{k}{2} - k$$

Finally, an edge in E' is made monochromatic (and thus, stored in Q) with probability p^{k-2} because $k-1$ different subsets must all be colored identically. This gives $\gamma = k-2$, so

$$\lambda = \min \left\{ \frac{2 \binom{k}{2} - k}{\binom{k}{2} - 1} - 1, \frac{k-2}{\binom{k}{2} - 1} \right\} = \frac{k-2}{\binom{k}{2} - 1} = \frac{2}{k+1}$$

for $k \geq 3$. Thus, our algorithm runs in $\tilde{O}(m/T^\lambda) = \tilde{O}(m/T^{2/(k+1)})$ bits of memory in the worst case. We have therefore proved the following result.

► **Theorem 16.** *There is a 2-pass algorithm that reads a stream of m hyperedges specifying a k -uniform hypergraph H and produces an $(\varepsilon, o(1))$ -estimate of $T_k(H)$; under the promise that $T_k(H) \geq T$, the algorithm runs in expected space $O(\varepsilon^{-2} \log^2 n \cdot m/T^{2/(k+1)})$. ◻*

5 Concluding Remarks

We initiated a systematic study of the simplex counting problem in a streamed hypergraph, a natural generalization of the much-studied triangle counting problem. We obtained several sublinear-space algorithms: which of them is best depends on some combinatorial parameters of the problem instance. Overall, we learned that established methods for triangle and substructure counting in graph streams are not by themselves enough and considerable effort is required to deal with the more intricate structures occurring in hypergraphs.

In some parameter regimes (what we called the “abundant” case), we obtained provably space-optimal algorithms. However, in the “meager” case, seeking algorithms with space complexity of the form m/T^λ , we established an upper bound with $\lambda = 2/(k+1)$ and a lower bound with $\lambda = 1 - 1/k$. Closing this gap seems to us to be the most compelling follow-up research question.

The simplex counting problem has the potential to impact research on pattern detection and enumeration. As triangle counting has done in the past two decades, simplex counting may offer new insights on how sampling techniques can exploit the structure of graphs and hypergraphs to extract meaningful information.



References

- 1 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. Preliminary version in *Proc. 2nd Annual European Symposium on Algorithms*, pages 354–364, 1994.
- 2 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In *Proc. 10th Conference on Innovations in Theoretical Computer Science*, volume 124 of *LIPICs*, pages 6:1–6:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 3 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002.
- 4 Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proc. 14th Annual SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 16–24, 2008.
- 5 Suman K. Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *Proc. 34th International Symposium on Theoretical Aspects of Computer Science*, pages 11:1–11:14, 2017.
- 6 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *Proc. 40th International Colloquium on Automata, Languages and Programming*, pages 244–254. Springer, 2013.
- 7 Laurent Bulteau, Vincent Froese, Konstantin Kutzkov, and Rasmus Pagh. Triangle counting in dynamic graph streams. *Algorithmica*, 76(1):259–278, 2016. doi:10.1007/s00453-015-0036-4.
- 8 Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proc. 25th ACM Symposium on Principles of Database Systems*, pages 253–262, 2006.
- 9 Amit Chakrabarti and Themistoklis Haris. Counting simplices in hypergraph streams. *arXiv preprint*, 2021. arXiv:2112.11016.
- 10 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.
- 11 Nicholas A Christakis and James H Fowler. Social network sensors for early detection of contagious outbreaks. *PLoS ONE*, 5(9):e12948, 2010.
- 12 Graham Cormode and Hossein Jowhari. A second look at counting triangles in graph streams. *Theor. Comput. Sci.*, 552:44–51, 2014.
- 13 Graham Cormode and Hossein Jowhari. A second look at counting triangles in graph streams (corrected). *Theor. Comput. Sci.*, 683:22–30, 2017.
- 14 Lech Duraj, Krzysztof Kleiner, Adam Polak, and Virginia Vassilevska Williams. Equivalences between triangle and range query problems. In Shuchi Chawla, editor, *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 30–47. SIAM, 2020.
- 15 Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 614–633, 2015.
- 16 Talya Eden, Dana Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. *SIAM J. Comput.*, 49(4):747–771, 2020.
- 17 David C. Fisher. Lower bounds on the number of triangles in a graph. *J. Graph Th.*, 13(4):505–512, 1989.
- 18 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018.
- 19 Chuangyi Gui, Long Zheng, Pengcheng Yao, Xiaofei Liao, and Hai Jin. Fast triangle counting on GPU. In *2019 IEEE High Performance Extreme Computing Conference, HPEC 2019, Waltham, MA, USA, September 24-26, 2019*, pages 1–7. IEEE, 2019.

- 20 Rajesh Jayaram and John Kallaugher. An optimal algorithm for triangle counting in the stream. In *Proc. 24th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 207 of *LIPICs*, pages 11:1–11:11, 2021.
- 21 Madhav Jha, C. Seshadhri, and Ali Pinar. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *ACM Trans. Knowl. Discov. Data*, 9(3):15:1–15:21, 2015.
- 22 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In Lusheng Wang, editor, *Proc. 11th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 710–716, 2005.
- 23 John Kallaugher, Michael Kapralov, and Eric Price. The sketching complexity of graph and hypergraph counting. In *Proc. 59th Annual IEEE Symposium on Foundations of Computer Science*, pages 556–567. IEEE Computer Society, 2018.
- 24 John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proc. 38th ACM Symposium on Principles of Database Systems*, pages 119–133. ACM, 2019.
- 25 John Kallaugher and Eric Price. A hybrid sampling scheme for triangle counting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1778–1797. SIAM, 2017.
- 26 Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Proc. 39th International Colloquium on Automata, Languages and Programming*, volume 7392 of *Lecture Notes in Computer Science*, pages 598–609. Springer, 2012.
- 27 Jeong Han Kim and Van H. Vu. Divide and conquer martingales and the number of triangles in a random graph. *Rand. Struct. Alg.*, 24(2):166–174, 2004.
- 28 Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, 8(1-2):161–185, 2012.
- 29 Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *Proc. 19th International Conference on World Wide Web (WWW)*, pages 641–650, 2010.
- 30 Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. Approximate counting of cycles in streams. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Proc. 19th Annual European Symposium on Algorithms*, volume 6942 of *Lecture Notes in Computer Science*, pages 677–688. Springer, 2011.
- 31 Willem Mantel. Problem 28 (solution by H. Gouweniak, W. Mantel, J. Texeira de Mattes, F. Schuh, and WA Whythoff). *Wiskundige Opgaven*, 10:60–61, 1907.
- 32 Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *Proc. 35th ACM Symposium on Principles of Database Systems*, pages 401–411, 2016.
- 33 Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.
- 34 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, 2018.
- 35 Rasmus Pagh and Charalampos E. Tsourakakis. Colorful triangle counting and a MapReduce implementation. *Information Processing Letters*, 112(7):277–281, 2012.
- 36 A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proc. VLDB Endowment*, 6(14):1870–1881, 2013.
- 37 Alexander A. Razborov. On the minimal density of triangles in graphs. *Comb. Probab. Comput.*, 17(4):603–618, 2008.

- 38 Jacques Rougemont and Pascal Hingamp. Dna microarray data and contextual analysis of correlation graphs. *BMC bioinformatics*, 4(1):1–11, 2003.
- 39 C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Wedge sampling for computing clustering coefficients and triangle counts on large graphs. *Stat. Anal. Data Min.*, 7(4):294–307, 2014.
- 40 Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proc. 20th International Conference on World Wide Web (WWW)*, pages 607–614. ACM, 2011.
- 41 Charalampos E. Tsourakakis, Mihail N. Kolountzakis, and Gary L. Miller. Triangle sparsifiers. *J. Graph Algorithms Appl.*, 15(6):703–726, 2011.
- 42 Sofya Vorotnikova. Improved 3-pass algorithm for counting 4-cycles in arbitrary order streaming. *CoRR*, abs/2007.13466, 2020. [arXiv:2007.13466](https://arxiv.org/abs/2007.13466).
- 43 Virginia Vassilevska Williams. 3sum and related problems in fine-grained complexity (invited talk). In *Proc. 37th ACM Symposium on Computational Geometry*, volume 189 of *LIPICs*, pages 2:1–2:2. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 44 Zhiang Wu, Jie Cao, Yaqiong Wang, Youquan Wang, Lu Zhang, and Junjie Wu. hPSD: a hybrid pu-learning-based spammer detection model for product reviews. *IEEE transactions on cybernetics*, 50(4):1595–1606, 2018.




Approximation Algorithms for Continuous Clustering and Facility Location Problems

Deeparnab Chakrabarty  

Department of Computer Science, Dartmouth College, Hanover, NH, USA

Maryam Negahbani  

Department of Computer Science, Dartmouth College, Hanover, NH, USA

Ankita Sarkar   

Department of Computer Science, Dartmouth College, Hanover, NH, USA

Abstract

In this paper, we consider center-based clustering problems where C , the set of points to be clustered, lies in a metric space (X, d) , and the set X of candidate centers is potentially infinite-sized. We call such problems *continuous* clustering problems to differentiate them from the *discrete* clustering problems where the set of candidate centers is explicitly given. It is known that for many objectives, when one restricts the set of centers to C itself and applies an α_{dis} -approximation algorithm for the discrete version, one obtains a $\beta \cdot \alpha_{\text{dis}}$ -approximation algorithm for the continuous version via the triangle inequality property of the distance function. Here β depends on the objective, and for many objectives such as k -median, $\beta = 2$, while for some others such as k -means, $\beta = 4$. The motivating question in this paper is *whether this gap of factor β between continuous and discrete problems is inherent, or can one design better algorithms for continuous clustering than simply reducing to the discrete case as mentioned above?* In a recent SODA 2021 paper, Cohen-Addad, Karthik, and Lee prove a factor-2 and a factor-4 hardness, respectively, for the continuous versions of the k -median and k -means problems, even when the number of cluster centers is a constant. The discrete problem for a constant number of centers is easily solvable exactly using enumeration, and therefore, in certain regimes, the “ β -factor loss” seems unavoidable.

In this paper, we describe a technique based on the *round-or-cut* framework to approach continuous clustering problems. We show that, for the continuous versions of some clustering problems, we can design approximation algorithms attaining a better factor than the β -factor blow-up mentioned above. In particular, we do so for: the uncapacitated facility location problem with uniform facility opening costs (λ -UFL); the k -means problem; the individually fair k -median problem; and the k -center with outliers problem. Notably, for λ -UFL, where $\beta = 2$ and the discrete version is NP-hard to approximate within a factor of 1.27, we describe a 2.32-approximation for the continuous version, and indeed $2.32 < 2 \times 1.27$. Also, for k -means, where $\beta = 4$ and the best known approximation factor for the discrete version is 9, we obtain a 32-approximation for the continuous version, which is better than $4 \times 9 = 36$.

The main challenge one faces is that most algorithms for the discrete clustering problems, including the state of the art solutions, depend on Linear Program (LP) relaxations that become infinite-sized in the continuous version. To overcome this, we design new linear program relaxations for the continuous clustering problems which, although having exponentially many constraints, are amenable to the round-or-cut framework.

2012 ACM Subject Classification Theory of computation \rightarrow Facility location and clustering

Keywords and phrases Approximation Algorithms, Clustering, Facility Location, Fairness, Outliers

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.33

Related Version *Full Version:* <https://arxiv.org/abs/2206.15105> [14]

Funding The authors were supported by NSF award #2041920.

Acknowledgements We thank the reviewers for their comments, which helped us improve our exposition and exhibit a stronger connection to existing work.



© Deeparnab Chakrabarty, Maryam Negahbani, and Ankita Sarkar; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 33; pp. 33:1–33:15

Leibniz International Proceedings in Informatics



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

1 Introduction

Clustering is a ubiquitous problem arising in various areas ranging from data analysis to operations research. One popular class of clustering problems are the so-called *center-based clustering* problems where the quality of the clustering is determined by a function of the distances of every point in C to the “centers” of the clusters they reside in. Two extensively studied measures are the sum of these distances, with the resulting problem called the k -median problem, and the sum of squares of these distances, with the resulting problem called the k -means problem.

In most settings, these center-based clustering problems are NP-hard and one considers approximation algorithms for the same. Traditionally, however, approximation algorithms for these problems have been studied in *finite/discrete* metric spaces and, in fact, usually under the constraint that the set of centers, aka facilities, can be selected only from a prescribed subset $F \subseteq X$ of the metric space. Indeed, this model makes perfect sense when considering applications in operations research where the possible depot-locations may be constrained. These discrete problems have been extensively studied [33, 45, 16, 30, 17, 15, 35, 34, 5, 37, 39, 40, 8] over the last three decades. For instance, for the k -median problem, the best known approximation algorithm is a 2.675-approximation [9], while the best known hardness is $(1 + 2/e) \approx 1.74$ [34]. For the k -means problem, the best known approximation algorithm is a $(9 + \varepsilon)$ -approximation algorithm [1, 31, 37], while the best hardness is $1 + 8/e \approx 3.94$ [34].

Restricting to a finite metric space, however, makes the problem easier, and indeed many of the above algorithms in the papers mentioned above would be infeasible to implement if X were extremely large – for instance, if X were \mathbb{R}^m for some large dimension m , and the distance function were the ℓ_p -metric, for some p . On the other hand, it is reasonably easy to show using triangle inequality that if one considers opening centers from C itself, and thus reduces the problem to its discrete version, then one incurs a hit of a factor β in the approximation factor, where β is a constant depending on the objective function. In particular, if we look at the sum-of-distances objectives such as in k -median, then $\beta = 2$, while if one looks at the sum-of-squared-distances such as in k -means, then $\beta = 4$. Therefore, one immediately gets a 5.35-approximation for the *continuous k -median* problem and a $36 + \varepsilon$ -approximation for the *continuous k -means* problem. The question we investigate is

Is this factor β hit necessary between the continuous and discrete versions of center based clustering problems, or can one design better approximation algorithms for the continuous case?

It is crucial to note that when considering designing algorithms, we do not wish to make *any* assumptions on the underlying metric space (X, d) . For instance, we do not wish to assume $X = \mathbb{R}^m$ for some m . This is important, for we really want to compare ourselves with the β which is obtained using only the triangle-inequality and symmetry property of d . On the other hand, to exhibit that a certain algorithm does not work, any candidate metric space suffices.

Recently, in a thought-provoking paper [21], Cohen-Addad, Karthik, and Lee show that, unless $\text{P} = \text{NP}$, the k -median and k -means problem defined on $(\mathbb{R}^m, \ell_\infty)$ cannot have an approximation ratio better than 2 and 4, respectively, *even when k is a constant!* Since the discrete problems have trivial exact algorithms via enumeration when k is a constant, this seems to indicate that in certain cases the above factor β hit is unavoidable. Is it possible that the *inapproximability* of the continuous problem is indeed β times the inapproximability of the discrete version?

1.1 Our Results

Our main contribution is a direct approach towards the continuous versions of clustering problems. We apply this to the following clustering problems where we obtain a factor better than $\beta \cdot \alpha_{\text{dis}}$, where α_{dis} is the best known factor for the discrete version of the problem.

- In the continuous λ -UFL problem, a “soft” version of the continuous k -median problem, one is allowed to pick any number of centers but has to pay a parameter λ for each picked center. The objective is to minimize the sum of distances of points in C to picked centers plus the cost for opening these centers. Again, note that the centers can be opened anywhere in X . For the discrete version, where the only possible center locations are in C , there is a 1.488-approximation due to Li [39], and a hardness of approximation within a factor of 1.278 is known due to Guha and Khuller [30]. We describe a 2.32-approximation algorithm. Note that $2.32 < 2 \cdot 1.278$ and thus, for this problem, the inapproximability is *not* β times that of the discrete case. We also show how the reduction of [21] carries over, to prove a hardness of 2 for this problem.
- In the continuous k -means problem, we wish to minimize the sum of squares of distances of clients to the closest open center. Recall that for this problem we have $\beta = 4$, and thus one gets a 36-factor algorithm for the continuous k -means using the best known 9-factor [1, 31, 37] algorithm for the discrete problem. We describe an improved 32-approximation for the continuous k -means problem.
- For the continuous k -median problem, our techniques fall short of improving $2\times$ the best known approximation factor for the discrete k -median problem. On the other hand, we obtain better algorithms for the the *individually fair* or *priority* version of the continuous k -median problem. In this problem, every point $v \in C$ has a specified radius $r(v)$ and desires a center opened within this distance. The objective is the same as the k -median problem: minimize the sum of all the distances. This problem arises as a possible model [36, 13, 41, 47, 43, 7] in the study of fair clustering solutions, since the usual k -median algorithms may place certain clients inordinately far away. At a technical level, this problem is a meld of the k -median and the k -center problems; the latter is NP-hard, which forces one to look at *bicriterion* approximations. An (α, γ) -approximation would return a solution within α times the optimum but may connect v to a point as far as $\gamma r(v)$ away. Again, any (α, γ) -approximation for the *discrete* version where $X = C$ would imply a $(2\alpha, 2\gamma)$ -approximation for the continuous version.

The best discrete approximation is (8, 3) due to Vakilian and Yalçınar [47] which would imply an (16, 3)-approximation for the continuous version. We describe an (8, 8)-approximation for the continuous version of the problem.

- In the k -center with outliers (kCwO) problem, we are given a parameter $m \leq |C|$, and we need to serve only m of the clients. The objective is the maximum distance of a served client to its center. The k -center objective is one of the objectives for which most existing discrete algorithms can compare themselves directly with the continuous optimum. The 3-approximation algorithm in [17] for the kCwO problem is one such example. However, the best known algorithm for kCwO for the discrete case (when $X = C$) is a 2-approximation by Chakrabarty, Goyal, and Krishnaswamy [11] which proceeds via LP rounding, and does not give a 2-approximation for continuous kCwO. This was explicitly noted in a work by Ding, Yu, and Wang (“... unclear if the resulting approximation ratio for the problem in Euclidean space.”) [26], that describes a 2-approximation for kCwO in Euclidean space, however, violating the number of clients served. We give a proper 2-approximation for the continuous kCwO problem (with no assumptions on the metric space) with no violations.

1.2 Our Technical Insight

Most state of the art approximation algorithms for center-based clustering problems are based on LP relaxations where one typically has variables y_i for *every* potential location of a center. When the set X is large, this approach becomes infeasible. Our main technical insight, underlying all our results, is to use a different style of linear program with polynomially many variables but exponentially many constraints. We then use the *round-or-cut* framework to obtain our approximation factor. More precisely, given a *potential* solution to our program, we either “round” it to get a desired solution within the asserted approximation factor, or we find a *separating hyperplane* proving that this potential solution is infeasible. Once this hyperplane is fed to the ellipsoid algorithm [29], the latter generates another potential solution, and the process continues. Due to the ellipsoid method’s guarantees, we obtain our approximation factor in polynomial time.

For every client $v \in C$, our LP relaxation has variables of the form $y(v, r)$, indicating whether there is some point $x \in X$ in an r -radius around v which is “open” as a center. Throughout the paper we use r as a quantity varying “continuously”, but it can easily be discretized, with a loss of at most $\frac{1}{\text{poly}(n)}$, to arise from a set of size $\leq \text{poly}(n)$. Thus there are only polynomially many such variables. We add the natural “monotonicity” constraints: $y(v, r) \leq y(v, s)$ whenever $r \leq s$. Interestingly, for one of the applications, we also need the monotonicity constraints for non-concentric balls: if $B(v, r) \subseteq B(u, s)$, then we need $y(v, r) \leq y(u, s)$.

We have a variable C_v indicating the cost the client v pays towards the optimal solution. Next, we connect the C_v ’s and the $y(v, r)$ ’s in the following ways (when $\beta = 1$, and something similar when $\beta = 2$). One connection states that for any r , $C_v \geq r \cdot (1 - y(v, r))$ and we add these to our LP. For the last two applications listed above, this suffices. However, one can also state the stronger condition of $C_v \geq \int_0^\infty (1 - y(v, r)) dr$. Indeed, the weaker constraint is the “Markov-style inequality” version of the stronger constraint.

Our second set of constraints restrict the $y(v, r)$ ’s to be “not too large”. For instance, for the fair k -median or kCwO problems where we are only allowed k points from X , we assert that for *any* set \mathcal{B} of disjoint balls $B(v, r)$, we must have the sum of the respective $y(v, r)$ ’s to be at most k . This set of constraints is exponentially many, and this is the set of constraints that need the round-or-cut machinery. For the λ -UFL problem, we have that the sum of the $y(v, r)$ ’s scaled by λ plus the sum of the C_v ’s should be at most opt_g , which is a running guess of opt .

Once we set up the framework above, then we can *port many existing rounding* algorithms for the discrete clustering problems without much hassle. In particular, this is true for rounding algorithms which use the C_v ’s as the core driving force. For the continuous λ -UFL problem, we port the rounding algorithm from the paper [45] by Shmoys, Tardos, and Aardal. For the continuous k -means problem, we port the rounding algorithm from the paper [16] by Charikar, Guha, Shmoys, and Tardos. For the continuous fair k -median problem, we port the rounding algorithm from the paper [13] by Chakrabarty and Negahbani, which itself builds on the algorithm present in the paper [2] by Alamdari and Shmoys. For the continuous kCwO problem, we port the rounding algorithm present in the paper [11] by Chakrabarty, Goyal, and Krishnaswamy.

Our results fall short for the continuous k -median problem (without fairness), where we can port the rounding algorithm from the paper [16] and get a 6.67-approximation. This, however, does not improve upon the 5.35-factor mentioned earlier.

1.3 Other Related Works and Discussion

The continuous k -means and median problems have been investigated quite a bit in the specific setting when $X = \mathbb{R}^m$ and when $d(u, v)$ is the ℓ_2 distance. The paper [42] by Matoušek describes an $(1 + \varepsilon)$ -approximation (PTAS) that runs in time $O(n \log^k n \cdot \varepsilon^{-O(k^2 m)})$. This led to a flurry of results [32, 25, 38, 18, 27] on obtaining PTASes with better dependencies on k and m via the applications of coresets. There is a huge and growing literature on coresets, and we refer the interested reader to the paper [24] by Cohen-Addad, Saulpic, and Schwiegelshohn, and the references within, for more information. Another approach to the continuous k -means problem has been local search. The paper [37] which describes a $9 + \varepsilon$ -approximation was first stated for the geometric setting, however it also went via the discretization due to Matoušek [42] and suffered a running time of exponential dependency on the dimension. More recent papers [28, 22] described local-search based PTASes for metrics with doubling dimension D , with running time exponentially depending on D . These doubling metrics generalize (\mathbb{R}^m, ℓ_2) -metrics. However, none of the above ideas seem to suggest better constant factor approximations for the continuous k -median/means problem in the general case, and indeed even when $X = \mathbb{R}^m$ but m is part of the input.

The k -means problem in the metric space (\mathbb{R}^m, ℓ_2) , where m and k are not constants, has been studied extensively [46, 6, 19, 23, 1, 20], and is called the Euclidean k -means problem. The discrete version of this problem was proved APX-hard in 2000 [46], but the APX-hardness of the continuous version was proved much later, in 2015 [6]. More recently, the hardness results for both versions have been improved: the discrete Euclidean k -means problem is hard to approximate to factor 1.17, while the continuous problem is hard to approximate to factor 1.07 [19]. Moreover, under assumption of a complexity theoretic hypothesis called the Johnson coverage hypothesis, these numbers have been improved to 1.73 and 1.36, respectively [23]. On the algorithmic side, the discrete Euclidean k -means problem admits a better approximation ratio than the general case: a 6.36 approximation was described in 2017 [1], which was very recently improved to 5.912 [20].

We believe that our paper takes the first stab at getting approximation ratios better than $\beta \times$ the best discrete factor for the *continuous* clustering problems. Round-or-cut is a versatile framework for approximation algorithm design with many recent applications [44, 10, 3, 12, 4], and the results in our paper is yet another application of this paradigm. However, many questions remain. We believe that the most interesting question to tackle is the continuous k -median problem. The best known discrete k -median algorithms are, in fact, combinatorial in nature, and are obtained via applying the primal-dual/dual-fitting based methods [35, 34, 40, 9] on the discrete LP. However, their application still needs an explicit description of the facility set, and it is interesting to see if they can be *directly* ported to the continuous setting.

All the algorithms in our paper, actually *still* open centers from C . Even then, we are able to do better than simply reducing to the discrete case, because we do not commit to the β loss upfront, and instead round from a fractional solution that can open centers anywhere in X . This raises an interesting question for the k -median problem (or any other center based clustering problem): consider the potentially infinite-sized LP which has variables y_i for all $i \in X$, but restrict to the optimal solution which only is allowed to open centers from C . How big is this “integrality gap”? It is not too hard to show that for the k -median problem this is between 2 and 4. The upper bound gives hope we can get a true 4-approximation for the continuous k -median problem, but it seems one would need new ideas to obtain such a result.

Organization of Extended Abstract

Due to space constraints, in this extended abstract we have decided to focus on the continuous λ -UFL and the continuous fair k -median results since we believe that they showcase the technical ideas in this paper. Proofs of certain statements have also been deferred to the full version of the paper [14]. The description of the results on continuous k -means and continuous k -center with outliers can be found in the full version [14, Appendices B and C].

2 Preliminaries

Given a metric space (X, d) on points X with pairwise distances d , we use the notation $d(v, S) = \min_{i \in S} d(v, i)$ for $v \in X$ and $S \subseteq X$ to denote v 's distance to the set S .

► **Definition 1** (Continuous k -median (Cont- k -Med)). *The input is a metric space (X, d) , clients $C \subseteq X$, $|C| = n \in \mathbb{N}$, and $k \in \mathbb{N}$. The goal is to find $S \subseteq X$, $|S| = k$ minimizing $\text{cost}(S) := \sum_{v \in C} d(v, S)$.*

► **Definition 2** (Continuous Fair k -median (ContFair- k -Med)). *Given the Cont- k -Med input, plus fairness radii $r : C \rightarrow [0, \infty)$, the goal is to find $S \subseteq X$, $|S| = k$ such that $\forall v \in C$, $d(v, S) \leq r(v)$, minimizing $\text{cost}(S)$.*

In the Uncapacitated Facility Location (UFL) problem, the restriction of opening only k facilities is replaced by having a cost associated with opening each facility. When these costs are equal to the same value λ for all facilities, the problem is called λ -UFL.

► **Definition 3** (Continuous λ -UFL (Cont- λ -UFL)). *Given a metric space (X, d) , clients $C \subseteq X$, $|C| = n \in \mathbb{N}$, and $\lambda \geq 0$, find $S \subseteq X$ that minimizes the sum of “connection cost” $\text{cost}_C(S) := \sum_{v \in C} d(v, S)$ and “facility opening cost” $\text{cost}_F(S) := \lambda|S|$.*

Let $\Delta = \max_{u, v \in C} d(u, v)$ denote the diameter of a metric (X, d) . For $x \in X$, $0 \leq r \leq \Delta$, the ball of radius r around x is $B(x, r) := \{x' \in X \mid d(x', x) \leq r\}$. Throughout the paper, we use balls of the form $B(v, r)$ where v is a client and $r \in \mathbb{R}$. To circumvent the potentially infinite number of radii, the radii can be discretized into $I_\varepsilon = \{\varepsilon, 2\varepsilon, \dots, \lceil \Delta/\varepsilon \rceil \varepsilon\}$ for a small constant $\varepsilon = O(1/n^2)$. Thereupon, we can appeal to the following lemma to bound the size of I_ε by $O(n^5)$.

► **Lemma 4** (Rewording of Lemma 4.1, [1]). *Losing a factor of $(1 + \frac{100}{n^2})$, we can assume that for any $u, v \in C$, $1 \leq d(u, v) \leq n^3$.*

For simplicity of exposition, we present our techniques using radii in \mathbb{R} , and observe that discretizing to I_ε incurs an additive loss of at most $O(n\varepsilon) = O(1/n)$ in our guarantees. We also note that $\log \text{opt} \leq \log(n\Delta) = O(\log n)$ by the above, which enables us to efficiently binary-search over our guesses opt_g .

3 Continuous λ -UFL

We start this section with our 2.32-approximation for Cont- λ -UFL (Theorem 5). For this, we introduce a new linear programming formulation, and adapt the rounding algorithm of Shmoys-Tardos-Aardal to the new program. The resulting procedure exhibits our main ideas, and serves as a warm-up for the remaining sections. Also, in Section 3.2, we prove that it is NP-hard to approximate Cont- λ -UFL within a factor of $2 - o(1)$, using ideas due to Cohen-Addad, Karthik, and Lee [21]. This shows that the continuous version cannot be approximated as well as the discrete version, which has a best-known approximation factor of 1.463 [39].

3.1 Approximation algorithm

This subsection is dedicated to proving the following theorem:

► **Theorem 5.** *There is a polynomial time algorithm that, for an instance of Cont- λ -UFL with optimum opt , yields a solution with cost at most $(\frac{2}{1-e^{-2}} + \varepsilon)\text{opt} < 2.32 \cdot \text{opt}$. Here $\varepsilon = O(\frac{1}{n^2})$.*

We design the following linear program for Cont- λ -UFL. We use variables C_v for the connection cost of each client v , and $y(v, r)$ for the number of facilities opened within each ball of the form $B(v, r)$. We also use a guess of the optimum opt_g , which we will soon discuss how to obtain. Throughout, we use $y(B)$ as shorthand for $y(v, r)$ where $B = B(v, r)$.

$$\lambda \sum_{B \in \mathcal{B}} y(B) + \sum_{v \in C} C_v \leq \text{opt}_g \quad \forall \mathcal{B} \subseteq \{B(v, r)\}_{\substack{v \in C \\ r \in \mathbb{R}}} \text{ pairwise disjoint} \quad (\text{UFL})$$

$$\int_0^\infty (1 - y(v, r)) dr \leq C_v \quad \forall v \in C, r \in \mathbb{R} \quad (\text{UFL-1})$$

$$y(v, r) \leq y(v, r') \quad \forall v \in C, r, r' \in \mathbb{R} \text{ s.t. } r \leq r' \quad (\text{UFL-2})$$

$$y(v, r) \geq 0, C_v \geq 0 \quad \forall v \in C, r \in \mathbb{R}$$

Observe that, given a solution $S \subseteq X$ of cost at most opt_g , we can obtain a feasible solution of UFL as follows. For client $v \in C$, we set $C_v = d(v, S)$. For $v \in C, r \in \mathbb{R}$, we set $y(v, r) = 0$ for $r < d(v, S)$ and $y(v, r) = 1$ for $r \geq d(v, S)$.

Our approach is to round a solution (C, y) of UFL. Observe that there are polynomially many constraints of the form (UFL-1) and (UFL-2); hence, we can efficiently obtain a solution (C, y) that satisfies them. So for the remainder of this section, we assume that those constraints are satisfied. On the other hand, there are infinitely many constraints of type (UFL). This is why we employ a round-or-cut framework via the ellipsoid algorithm [29]. We begin with an arbitrary opt_g , and when ellipsoid asks us if a proposed solution (C, y) is feasible, we run the following algorithm.

The algorithm inputs $\alpha < 1$, and defines $r_\alpha(v)$ as the minimum radius at which client $v \in C$ has at least α mass of open facilities around it. First, all clients are deemed *uncovered* ($U = C$). Iteratively, the algorithm picks the j , i.e. the uncovered client, with the smallest $r_\alpha(j)$. j is put into the set $\text{Reps}_{(C, y), \alpha}$. Any client v within distance $r_\alpha(j) + r_\alpha(v)$ of j is considered a child of j and is now *covered*. When all clients are covered, i.e. $U = \emptyset$, the algorithm outputs $\text{Reps}_{(C, y), \alpha}$.

■ **Algorithm 1** Filtering for Cont- λ -UFL.

Input: A proposed solution $(\{C_v\}_{v \in C}, \{y(v, r)\}_{v \in C, r \in \mathbb{R}})$ for UFL, parameter $\alpha \in (e^{-2}, 1)$

- 1: $r_\alpha(v) \leftarrow \min\{r \in \mathbb{R} \mid y(v, r) \geq \alpha\}$ for all $v \in C$
- 2: $\text{Reps}_{(C, y), \alpha} \leftarrow \emptyset$ ▷ “representative” clients
- 3: $U \leftarrow C$ ▷ “uncovered” clients
- 4: **while** $U \neq \emptyset$ **do**
- 5: Pick $j \in U$ with minimum $r_\alpha(j)$
- 6: $\text{child}(j) \leftarrow \{v \in U \mid d(v, j) \leq r_\alpha(v) + r_\alpha(j)\}$
- 7: $U \leftarrow U \setminus \text{child}(j)$
- 8: $\text{Reps}_{(C, y), \alpha} \leftarrow \text{Reps}_{(C, y), \alpha} \cup j$
- 9: **end while**

Output: $\text{Reps}_{(C, y), \alpha}$

Notice that, by construction, the collection of balls $\{B(j, r_\alpha(j))\}_{j \in \text{Reps}_{(C,y),\alpha}}$ is pairwise disjoint. Hence, the following constraint, which we call $\text{Sep}_{(C,y),\alpha}$, is of the form (UFL):

$$\lambda \sum_{j \in \text{Reps}_{(C,y),\alpha}} y(j, r_\alpha(j)) + \sum_{v \in C} C_v \leq \text{opt}_g \quad (\text{Sep}_{(C,y),\alpha})$$

We will show that

► **Lemma 6.** *If (C, y) satisfies (UFL-1), (UFL-2), and $\text{Sep}_{(C,y),\alpha}$, then there exists a suitable $\alpha \in (e^{-2}, 1)$ for which the output of Algorithm 1 has cost at most $\frac{2}{1-e^{-2}} \text{opt}_g$.*

Thus, if we find that the desired approximation ratio is not attained, then it must be that $\text{Sep}_{(C,y),\alpha}$ was not satisfied, and we can pass it to ellipsoid as a separating hyperplane. If ellipsoid finds that the feasible region of our linear program is empty, then we increase opt_g and try again. Otherwise, we obtain a solution $\text{Reps}_{(C,y),\alpha}$ that attains the desired guarantees.

We now analyze Algorithm 1 to prove Lemma 6.

Proof of Lemma 6. For this proof, we will fix (C, y) , and refer to $\text{Reps}_{(C,y),\alpha}$ as Reps_α .

To prove a suitable α exists, assume α is picked uniformly at random from $(\beta, 1)$ for some $0 < \beta < 1$; we will see later that $\beta = e^{-2}$ is optimal. Take Reps_α , the output of Algorithm 1 on (C, y) . By definition of r_α , $\sum_{j \in \text{Reps}_\alpha} y(j, r_\alpha(j)) \geq \alpha |\text{Reps}_\alpha|$. Thus $\text{cost}_F(\text{Reps}_\alpha) = \lambda |\text{Reps}_\alpha| \leq \frac{1}{\alpha} \cdot \lambda \sum_{j \in \text{Reps}_\alpha} y(j, r_\alpha(j))$, which implies

$$\mathbf{Exp}[\text{cost}_F(\text{Reps}_\alpha)] \leq \frac{\ln(1/\beta)}{(1-\beta)} \lambda \sum_{j \in \text{Reps}_\alpha} y(j, r_\alpha(j)). \quad (1)$$

To bound the expected connection cost, take $v \in C$ and observe that, since all the clients are ultimately covered in Algorithm 1, there has to exist $j \in \text{Reps}_\alpha$ for which $v \in \text{child}(j)$. By construction of child , $d(v, j) \leq r_\alpha(v) + r_\alpha(j)$, which is at most $2r_\alpha(v)$ by our choice of j in Line 5. Thus, for any client v , we get $d(v, \text{Reps}_\alpha) \leq d(v, j) \leq 2r_\alpha(v)$. So we are left to bound $\mathbf{Exp}[r_\alpha(v)]$ for an arbitrary client $v \in C$.

We have that

$$\mathbf{Exp}[r_\alpha(v)] = \frac{1}{1-\beta} \int_\beta^1 r_\alpha(v) d\alpha \leq \frac{1}{1-\beta} \int_0^1 r_\alpha(v) d\alpha.$$

We notice that at $\alpha = y(v, r)$, $r_\alpha(v) = r$. Also, $r_0(v) = 0$. So given (UFL-2) for all balls $B(v, r)$ with $r \in \mathbb{R}$, we can apply a change of variable to the integral to get $\int_0^1 r_\alpha(v) d\alpha = \int_{r_0(v)}^{r_1(v)} (y(v, r_1(v)) - y(v, r)) dr \leq \int_0^\infty (1 - y(v, r)) dr \leq C_v$, where the last inequality is by (UFL-1). Thus we have $\mathbf{Exp}[r_\alpha(v)] \leq \frac{C_v}{1-\beta}$. Summing $d(v, \text{Reps}_\alpha)$ over all $v \in C$ we have

$$\mathbf{Exp}[\text{cost}_C(\text{Reps}_\alpha)] = \sum_{v \in C} \mathbf{Exp}[d(v, \text{Reps}_\alpha)] \leq 2 \sum_{v \in C} \mathbf{Exp}[r_\alpha(v)] \leq \frac{2}{1-\beta} \sum_{v \in C} C_v. \quad (2)$$

To balance cost_F from (1) and cost_C from (2), we set $\beta = e^{-2}$. The expected Cont- λ -UFL cost of Reps_α is, using $\text{Sep}_{(C,y),\alpha}$,

$$\mathbf{Exp}[\text{cost}_F(\text{Reps}_\alpha) + \text{cost}_C(\text{Reps}_\alpha)] \leq \frac{2}{1-e^{-2}} \left(\lambda \sum_{j \in \text{Reps}_\alpha} y(j, r_\alpha(j)) + \sum_{v \in C} C_v \right) \leq \frac{2 \cdot \text{opt}_g}{1-e^{-2}}.$$

Since the bound holds in expectation over a random α , there must exist an $\alpha \in (\beta, 1)$ that satisfies it deterministically. ◀

To obtain a suitable α , we can adapt the derandomization procedure from the discrete version [45]. The procedure relies on having polynomially many interesting radii; for this, we recall that while we have used $r \in \mathbb{R}$ for simplicity, our radii are actually $r \in I_\varepsilon$, $|I_\varepsilon| = O(n^5)$.

3.2 Hardness of approximation

Our hardness result for this problem is as follows:

► **Theorem 7.** *Given an instance of Cont- λ -UFL and $\varepsilon > 0$, it is NP-hard to distinguish between the following:*

- *There exists $S \subseteq X$ such that $\text{cost}_F(S) + \text{cost}_C(S) \leq (1 + 6\varepsilon)n$*
- *For any $S \subseteq X$, $\text{cost}_F(S) + \text{cost}_C(S) \geq (2 - \varepsilon)n$*

Thus we exhibit hardness of approximation up to a factor of $\frac{2-\varepsilon}{1+6\varepsilon}$, which tends to 2 as $\varepsilon \rightarrow 0$. Our reduction closely follows the hardness proof for Cont- k -Med [21]. The details appear in the full version of this paper [14, Appendix D].

4 Continuous Fair k -Median

The main result of this section is the following theorem.

► **Theorem 8.** *There exists a polynomial time algorithm for ContFair- k -Med that, for an instance with optimum cost opt , yields a solution with cost at most $8\text{opt} + \varepsilon$, in which, each client $v \in C$ is provided an open facility within distance $8r(v) + \varepsilon$ of itself. Here $\varepsilon = O\left(\frac{1}{n^2}\right)$.*

We create a round-or-cut framework, via the ellipsoid algorithm [29], that adapts the Chakrabarty-Negahbani algorithm [13] to the continuous setting. For this, we will modify the UFL linear program to suit ContFair- k -Med. As before, opt_g is a guessed optimum, C_v is the cost share of a client $v \in C$, and $y(v, r)$ represents the number of facilities opened in $B(v, r)$. There are two key modifications. First, we expand the monotonicity constraints of the form (UFL-2) to include *non-concentric* balls, which are crucial for adapting the fairness guarantee of Chakrabarty and Negahbani [13]. Second, we enforce the fairness constraints by requiring $y(v, r(v)) \geq 1$ for each client $v \in C$.

$$\sum_{v \in C} C_v \leq \text{opt}_g \quad (\text{LP})$$

$$\sum_{B \in \mathcal{B}} y(B) \leq k \quad \forall \mathcal{B} \subseteq \{B(v, r)\}_{\substack{v \in C \\ r \in \mathbb{R}}} \text{ pairwise disjoint} \quad (\text{LP-1})$$

$$\int_0^\infty (1 - y(v, r)) dr \leq C_v \quad \forall v \in C \quad (\text{LP-2})$$

$$y(u, r) \leq y(v, r') \quad \forall u, v \in C, r, r' \in \mathbb{R}, B(u, r) \subseteq B(v, r') \quad (\text{LP-3})$$

$$y(v, r(v)) \geq 1 \quad \forall v \in C \quad (\text{LP-4})$$

$$y(v, r) \geq 0, C_v \geq 0 \quad \forall v \in C, r \in \mathbb{R}$$

We will frequently use the following property of LP. See the full version of our paper [14, Appendix A.2] for the proof.

► **Lemma 9.** *Consider a solution (C, y) of LP. If for a client v , (C, y) satisfies all constraints of the form (LP-2) and (LP-3) involving v , then for any $r_0 \in \mathbb{R}$, $C_v \geq r_0(1 - y(v, r_0))$.*

As before, we will only worry about the constraints that are exponentially many. These are (LP-1). For this, we use ellipsoid [29]. Given a proposed solution (C, y) of LP, we construct $\text{Reps}_{(C,y)} \subseteq C$, as follows.

We first perform a filtering step. For each $v \in C$, we define $R(v) := \min r(v), 2C_v$. In the beginning, all clients are “uncovered” (i.e. $U = C$). In each iteration, let $j \in U$ be the uncovered client with the minimum $R(j)$; and add j to our set of “representatives” $\text{Reps}_{(C,y)}$. Any $v \in U$ within distance $2R(v)$ of j (including j itself) will be added to the set $\text{child}(j)$, and will be removed from U . After all clients are covered, i.e. $U = \emptyset$, the algorithm outputs $\text{Reps}_{(C,y)}$. For a formal description of this algorithm, see the full version [14, Appendix A.1].

For a $j \in \text{Reps}_{(C,y)}$, let $s(j)$ be the closest client to j in $\text{Reps}_{(C,y)} \setminus \{j\}$. Let $a(j) := \frac{d(j,s(j))}{2}$. So the collection of balls $\{B(j, a(j))\}_{j \in \text{Reps}_{(C,y)}}$ is pairwise disjoint, and the following constraint, which we call $\text{Sep}_{(C,y)}$, is of the form (LP-1).

$$\sum_{j \in \text{Reps}_{(C,y)}} y(j, a(j)) \leq k \quad (\text{Sep}_{(C,y)})$$

We have that

► **Lemma 10.** *If (C, y) satisfies (LP-2)-(LP-4) and $\text{Sep}_{(C,y)}$, then*

1. $\forall j \in \text{Reps}_{(C,y)}, y(j, a(j)) \geq \frac{1}{2}$
2. $|\text{Reps}_{(C,y)}| \leq 2k$

Proof. Fix $j \in \text{Reps}_{(C,y)}$. By construction of $\text{Reps}_{(C,y)}$,

$$a(j) = \frac{d(j, s(j))}{2} \geq R(j) \quad (3)$$

So if $R(j) = r(j)$, then by (LP-3) and (LP-4), $y(j, a(j)) \geq y(j, r(j)) \geq 1$. Else $R(j) = 2C_j$. By Lemma 9, $C_j \geq a(j)(1 - y(j, a(j)))$.

If $C_j = 0$, then this implies $y(j, a(j)) \geq 1$. Otherwise, substituting $a(j)$ by $R(j)$ from (3) and setting $R(j) = 2C_j$ gives $C_j \geq 2C_j(1 - y(j, a(j)))$, i.e. $y(j, a(j)) \geq \frac{1}{2}$. Now, by $\text{Sep}_{(C,y)}$, we have $k \geq \sum_{j \in \text{Reps}_{(C,y)}} y(j, a(j)) \geq \frac{1}{2} \cdot |\text{Reps}_{(C,y)}|$. ◀

So if we find that $|\text{Reps}_{(C,y)}| > 2k$, then $\text{Sep}_{(C,y)}$ must be violated, and we can pass it to ellipsoid as a separating hyperplane. Hence in polynomial time, we either find that our feasible region is empty, or we get (C, y) and $\text{Reps}_{(C,y)}$ such that (C, y) satisfies (LP), (LP-2)-(LP-4), and $\text{Sep}_{(C,y)}$. In the first case, we increase opt_g and try again. In the latter case, we round (C, y) further to attain our desired approximation ratios, via a rounding algorithm that we will now describe. This algorithm focuses on $\text{Reps}_{(C,y)}$ and ignores other clients, as justified by the following lemma.

► **Lemma 11.** *$S \subseteq X$ be a solution to ContFair- k -Med. Consider a proposed solution (C, y) of LP that satisfies (LP). Then $\sum_{v \in C} d(v, S) \leq \sum_{j \in \text{Reps}_{(C,y)}} |\text{child}(j)| d(j, S) + 4\text{opt}_g$.*

The proof closely follows from a standard technique for the discrete version [16, 13]. We provide the proof in the full version [14, Appendix A.3].

Our algorithm will also ignore facilities outside $\text{Reps}_{(C,y)}$, so our solution will be a subset of $\text{Reps}_{(C,y)}$. For the remainder of this section, we fix (C, y) , and refer to $\text{Reps}_{(C,y)}$ as Reps . We write the following polynomial-sized linear program, DLP, where Reps are the only clients and the only facilities. The objective function of DLP is a lower bound on $\sum_{j \in \text{Reps}} |\text{child}(j)| d(j, S)$, so hereafter we compare our output with DLP. We do not include fairness constraints in this program, and we will see later that it is not necessary to do so.

In DLP, the variables z_i for each $i \in \text{Reps}$ denote whether i is open as a facility. The variables x_{ij} for $i, j \in \text{Reps}$ denote whether the client j uses the facility i .

$$\text{minimize } \sum_{j \in \text{Reps}} |\text{child}(j)| \sum_{i \in \text{Reps}} x_{ij} d(j, i) \quad (\text{DLP})$$

$$\sum_{i \in \text{Reps}} z_i \leq k \quad (\text{DLP-1})$$

$$\sum_{i \in \text{Reps}} x_{ij} = 1 \quad \forall j \in \text{Reps} \quad (\text{DLP-2})$$

$$x_{ij} \leq y_i \quad \forall i, j \in \text{Reps} \quad (\text{DLP-3})$$

$$x_{ij} \geq 0, z_i \geq 0 \quad \forall i, j \in \text{Reps}$$

We will now round (C, y) to an integral solution of DLP. Our first step is to convert (C, y) to a fractional solution (\bar{x}, \bar{z}) of DLP. To do this, for each $j \in \text{Reps}$, we consolidate the y -mass in $B(j, a(j))$ onto j , i.e. we set $\bar{z}_j = y(j, a(j))$. By Lemma 10.1, each \bar{z}_j is then at least $\frac{1}{2}$. This allows j to use only itself and $s(j)$ as its fractional facilities.

■ **Algorithm 2** Consolidation for ContFair- k -Med.

Input: A proposed solution $(\{C_v\}_{v \in C}, \{y(v, r)\}_{v \in C, r \in I_\epsilon})$ for LP, and Reps as defined above

- 1: **for** $j \in \text{Reps}$ **do**
- 2: $s(j) \leftarrow \arg \min_{v \in \text{Reps} \setminus j} d(j, v)$
- 3: $a(j) \leftarrow d(j, s(j))/2$
- 4: $\bar{z}_j \leftarrow \min\{y(j, a(j)), 1\}$
- 5: $\bar{x}_{jj} \leftarrow \bar{z}_j$
- 6: $\bar{x}_{js(j)} \leftarrow 1 - \bar{z}_j$
- 7: **end for**

Output: (\bar{x}, \bar{z})

► **Lemma 12.** $\forall j \in \text{Reps}, \bar{z}_j \geq \frac{1}{2}$, and (\bar{x}, \bar{z}) is a feasible solution of DLP with cost at most 2opt_g .

Proof. For a $j \in \text{Reps}$, if $y(j, a(j)) = 1$ then $\bar{z}_j = 1$. Otherwise, by Lemma 10.1, $\bar{z}_j = y(j, a(j)) \geq \frac{1}{2}$.

Hence $1 - \bar{z}_j \leq \frac{1}{2} \leq \bar{z}_{s(j)}$, which implies feasibility by construction and $\text{Sep}_{(C, y)}$. It also implies that $\sum_{i \in \text{Reps}} \bar{x}_{ij} d(j, i) = (1 - \bar{z}_j) d(j, s(j))$. If $y(j, a(j)) = 1$, then the RHS above is $0 \leq 2C_j$. Otherwise $\sum_{i \in \text{Reps}} \bar{x}_{ij} d(j, i) = (1 - y(j, a(j))) d(j, s(j)) = 2(1 - y(j, a(j))) a(j) \leq 2C_j$ where the last inequality follows from Lemma 9. Multiplying by $|\text{child}(j)|$ and summing over all $j \in \text{Reps}$, we have by construction of Reps ,

$$\sum_{j \in \text{Reps}} |\text{child}(j)| \sum_{i \in \text{Reps}} \bar{x}_{ij} d(j, i) \leq 2 \sum_{j \in \text{Reps}} |\text{child}(j)| C_j \leq 2 \sum_{j \in \text{Reps}} \sum_{v \in \text{child}(j)} C_v = 2 \sum_{v \in C} C_v$$

which is at most 2opt_g by (LP). ◀

Now, to round (\bar{x}, \bar{z}) to an integral solution, we appeal to an existing technique [16, 13]. We state the relevant result here, and provide the proof in the full version [14, Appendix A.4].

► **Lemma 13** ([16, 13]). *Let (\bar{x}, \bar{z}) be a feasible solution of DLP with cost at most 2opt_g , such that $\forall j \in \text{Reps}, \bar{z}_j \geq \frac{1}{2}$. Then there exists a polynomial time algorithm that produces $S \subseteq \text{Reps}$ such that*

1. $|S| = k$;
2. If $\bar{z}_j = 1$, then $j \in S$;
3. $\forall j \in \text{Reps}$, at least one of $j, s(j)$ is in S ; and
4. $\sum_{j \in \text{Reps}} |\text{child}(j)| d(j, S) \leq 4\text{opt}_g$.

Thus, by Lemma 11, we have shown that $\sum_{v \in C} d(v, S) \leq \sum_{j \in \text{Reps}} |\text{child}(j)| \sum_{i \in S} d(j, S) + 4\text{opt}_g \leq 8\text{opt}_g$. Now we show the fairness ratio, adapting a related result [13, Lemma 3] from the discrete version. This is where we crucially require the monotonicity constraints (LP-3) for *non-concentric* balls.

► **Lemma 14.** $\forall v \in C, d(v, S) \leq 8r(v)$.

Proof. Fix $v \in C$, and let $j \in \text{Reps}$ such that $v \in \text{child}(j)$. By construction of $\text{child}(j)$,

$$d(v, j) \leq 2R(v) \leq 2r(v) \tag{4}$$

So if $j \in S$, then we are done. Otherwise, by Lemma 13.2, $\bar{z}_j < 1$, i.e. by Algorithm 2, $y(j, a(j)) < 1$. But by the fairness constraints (LP-4), $y(v, r(v)) \geq 1$. So by the monotonicity constraints (LP-3), $B(v, r(v)) \not\subseteq B(j, a(j))$, as otherwise we would have $1 \leq y(v, r(v)) \leq y(j, a(j)) < 1$, a contradiction.

So fix $w \in B(v, r(v)) \setminus B(j, a(j))$. We have

$$a(j) < d(j, w), \quad d(v, w) \leq r(v) \tag{5}$$

By Lemma 13.3, either $j \in S$ or $s(j) \in S$, so

$$\begin{aligned} d(j, S) &\leq d(j, s(j)) = 2a(j) < 2d(j, w) && \dots \text{ by (5)} \\ &\leq 2(d(v, j) + d(v, w)) \leq 2(2r(v) + r(v)) && \dots \text{ by (4) and (5)} \\ &= 6r(v) \end{aligned}$$

So, by (4), $d(v, S) \leq d(v, j) + d(j, S) \leq 2r(v) + 6r(v) = 8r(v)$. ◀

Thus we have proved Theorem 8.

We observe here that, by the simple reduction of setting all $r(v)$'s to ∞ , Theorem 8 implies a solution of cost $8\text{opt} + \varepsilon$ for *Cont- k -Med*. We improve this ratio via an improved rounding procedure by Charikar, Guha, Shmoys, and Tardos [16], which rounds (\bar{x}, \bar{z}) such that $\sum_{j \in \text{Reps}} |\text{child}(j)| d(j, S) \leq \frac{8}{3}\text{opt}_g$, instead of the 4opt_g that we obtain above. This yields:

► **Corollary 15.** *There exists a polynomial time algorithm for Cont- k -Med that, on an instance with optimum cost opt , yields a solution of cost at most $6\frac{2}{3}\text{opt} + \varepsilon$.*

This improved rounding, however, no longer guarantees to open either j or $s(j)$ for each $j \in \text{Reps}$. Such a guarantee (Lemma 13.3) is crucial to our fairness bound in Lemma 14. So the improvement is not naively adaptable to *ContFair- k -Med*.

References


- 1 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k -means and euclidean k -median by primal-dual algorithms. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72, 2017.
- 2 Soroush Alamdari and David B. Shmoys. A bicriteria approximation algorithm for the k -center and k -median problems. In *Proc., Workshop on Approximation and Online Algorithms (WAOA)*, pages 66–75. Springer, 2017.

- 3 Hyung-Chan An, Mohit Singh, and Ola Svensson. LP-based algorithms for capacitated facility location. *SIAM Journal on Computing (SICOMP)*, 46(1):272–306, 2017. Preliminary version in FOCS 2014.
- 4 Georg Anegg, Haris Angelidakis, Adam Kurpisz, and Rico Zenklusen. A technique for obtaining true approximations for k-center with covering constraints. *Math. Program.*, 192(1):3–27, 2022. doi:10.1007/s10107-021-01645-y.
- 5 V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local Search Heuristics for k-Median and Facility Location Problems. *SIAM Journal on Computing (SICOMP)*, 33(3):544–562, 2004.
- 6 Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of euclidean k-means. *CoRR*, 2015. arXiv:1502.03316.
- 7 Tanvi Bajpai, Deeparnab Chakrabarty, Chandra Chekuri, and Maryam Negahbani. Revisiting Priority k-Center: Fairness and Outliers. In *Proc., International Colloquium on Automata, Languages and Programming (ICALP)*, pages 21:1–21:20, 2021.
- 8 Jaroslav Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM*, 60(1):1–33, 2013. doi:10.1145/2432622.2432628.
- 9 Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 2017.
- 10 Deeparnab Chakrabarty, Chandra Chekuri, Sanjeev Khanna, and Nitish Korula. Approximability of capacitated network design. *Algorithmica*, 72(2):493–514, 2015.
- 11 Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The non-uniform k-center problem. *ACM Trans. Algorithms*, 2020. Preliminary version in ICALP 2016.
- 12 Deeparnab Chakrabarty and Maryam Negahbani. Generalized center problems with outliers. *ACM Trans. Algorithms*, 2019.
- 13 Deeparnab Chakrabarty and Maryam Negahbani. Better algorithms for individually fair k-clustering. *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, 2021.
- 14 Deeparnab Chakrabarty, Maryam Negahbani, and Ankita Sarkar. Approximation algorithms for continuous clustering and facility location problems. *CoRR*, 2022. arXiv:2206.15105.
- 15 Moses Charikar and Sudipto Guha. Improved Combinatorial Algorithms for the Facility Location and k-Median Problems. In *Proc., IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.
- 16 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.
- 17 Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 642–651, 2001.
- 18 Ke Chen. On k-median clustering in high dimensions. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1177–1185, 2006.
- 19 Vincent Cohen-Addad and Karthik C.S. Inapproximability of clustering in lp metrics. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 519–539, 2019. doi:10.1109/FOCS.2019.00040.
- 20 Vincent Cohen-Addad, Hossein Esfandiari, Vahab Mirrokni, and Shyam Narayanan. Improved approximations for Euclidean k-means and k-median, via nested quasi-independent sets. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, pages 1621–1628, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520011.
- 21 Vincent Cohen-Addad, C. S. Karthik, and Euiwoong Lee. On approximability of clustering problems without candidate centers. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2635–2648, 2021.





- 22 Vincent Cohen-Addad, Philip N Klein, and Claire Mathieu. Local search yields approximation schemes for k -means and k -median in euclidean and minor-free metrics. *SIAM Journal on Computing (SICOMP)*, 48(2):644–667, 2019.
- 23 Vincent Cohen-Addad, Karthik C. S., and Euiwoong Lee. Johnson coverage hypothesis: Inapproximability of k -means and k -median in ℓ_p -metrics. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1493–1530, 2022. doi:10.1137/1.9781611977073.63.
- 24 Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coresets framework for clustering. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 169–182, 2021.
- 25 W Fernandez De La Vega, Marek Karpinski, Claire Kenyon, and Yuval Rabani. Approximation schemes for clustering problems. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 50–58, 2003.
- 26 Hu Ding, Haikuo Yu, and Zixiu Wang. Greedy strategy works for k -center clustering with outliers and coresets construction. In *Proc., European Symposium on Algorithms*, pages 40:1–40:16, 2019.
- 27 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k -means clustering based on weak coresets. In *Proceedings of the 5th Annual Symposium on Computational Geometry (SoCG)*, pages 11–18, 2007.
- 28 Zachary Friggstad, Mohsen Rezapour, and Mohammad R Salavatipour. Local search yields a PTAS for k -means in doubling metrics. *SIAM Journal on Computing (SICOMP)*, 48(2):452–480, 2019.
- 29 Martin Grötschel, László Lovász, and Alexander Schriver. *Geometric Algorithms and Combinatorial Optimization*. Springer, Berlin, Heidelberg, 1993.
- 30 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.
- 31 Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *CoRR*, 2008. arXiv:0809.2554.
- 32 S. Har-Peled and S. Mazumdar. Coresets for k -means and k -median clustering and their applications. In *Proc., ACM Symposium on Theory of Computing (STOC)*, pages 291–300, 2004.
- 33 Dorit S. Hochbaum and David B. Shmoys. A best possible heuristic for the k -center problem. *Math. Oper. Res.*, 10(2):180–184, 1985. doi:10.1287/moor.10.2.180.
- 34 Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003.
- 35 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k -median problems using the primal-dual schema and Lagrangean relaxation. *Journal of the ACM*, 48(2):274–296, 2001.
- 36 Christopher Jung, Sampath Kannan, and Neil Lutz. Service in your neighborhood: Fairness in center location. In *Proceedings, Foundations of Responsible Computing, FORC 2020*, volume 156, pages 5:1–5:15, 2020.
- 37 Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. A local search approximation algorithm for k -means clustering. *Computational Geometry*, 28(2-3):89–112, 2004.
- 38 A. Kumar, Y. Sabharwal, and S. Sen. A simple linear time $(1 + \epsilon)$ -approximation algorithm for k -means clustering in any dimensions. In *Proc., IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 454–462, 2004.
- 39 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013.
- 40 Shi Li and Ola Svensson. Approximating k -median via pseudo-approximation. *SIAM Journal on Computing (SICOMP)*, 45(2):530–547, 2016.

- 41 Sepideh Mahabadi and Ali Vakilian. (individual) fairness for k -clustering. In *Proc., International Conference on Machine Learning (ICML)*, pages 7925–7935, 2020.
- 42 Jiri Matoušek. On approximate geometric k -clustering. *Discrete & Computational Geometry*, 24(1):61–84, 2000.
- 43 Ján Plesník. A heuristic for the p -center problems in graphs. *Discrete Applied Mathematics*, 17(3):263–268, 1987.
- 44 Robert D. Carr, Toshihiro Fujito, Goran Konjevod, and Ojas Parekh. A $2\frac{1}{10}$ -approximation Algorithm for a Generalization of the Weighted Edge-Dominating Set Problem. *J. Comb. Optim.*, 2001. Preliminary version appeared in *Proc. 8th ESA*, pages 132–142, 2000.
- 45 David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proc., ACM Symposium on Theory of Computing (STOC)*, 1997.
- 46 Luca Trevisan. When hamming meets euclid: The approximability of geometric tsp and steiner tree. *SIAM Journal on Computing*, 30(2):475–485, 2000. doi:10.1137/S0097539799352735.
- 47 Ali Vakilian and Mustafa Yalciner. Improved approximation algorithms for individually fair clustering. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 8758–8779. PMLR, 28–30 March 2022. URL: <https://proceedings.mlr.press/v151/vakilian22a.html>.


Distinct Elements in Streams: An Algorithm for the (Text) Book

Sourav Chakraborty ¹   

Indian Statistical Institute Kolkata, India

N. V. Vinodchandran ¹   

University of Nebraska-Lincoln, NE, USA

Kuldeep S. Meel   

National University of Singapore, Singapore

Abstract

Given a data stream $\mathcal{D} = \langle a_1, a_2, \dots, a_m \rangle$ of m elements where each $a_i \in [n]$, the Distinct Elements problem is to estimate the number of distinct elements in \mathcal{D} . Distinct Elements has been a subject of theoretical and empirical investigations over the past four decades resulting in space optimal algorithms for it. All the current state-of-the-art algorithms are, however, beyond the reach of an undergraduate textbook owing to their reliance on the usage of notions such as pairwise independence and universal hash functions. We present a simple, intuitive, sampling-based space-efficient algorithm whose description and the proof are accessible to undergraduates with the knowledge of basic probability theory.

2012 ACM Subject Classification Theory of computation \rightarrow Sketching and sampling

Keywords and phrases F_0 Estimation, Streaming, Sampling

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.34

Funding This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme (NRF-NRFFAI1-2019-0004), Singapore Ministry of Education Academic Research Fund Tier 1, Ministry of Education Singapore Tier 2 grant (MOE-T2EP20121-0011), and Amazon Faculty Research Awards. Vinod was supported in part by NSF CCF-2130608 and NSF HDR:TRIPODS-1934884 awards.


1 Introduction

We consider the fundamental problem of estimating the number of distinct elements in a data stream (Distinct Elements problem or the F_0 estimation problem). For a data stream $\mathcal{D} = \langle a_1, a_2, \dots, a_m \rangle$, where each $a_i \in [n]$, $F_0(\mathcal{D})$ is the number of distinct elements in \mathcal{D} : $F_0(\mathcal{D}) = |\{a_1, a_2, \dots, a_m\}|$.

► **Problem 1.** *Given a stream $\mathcal{D} = \langle a_1, a_2, \dots, a_m \rangle$ of m elements where each $a_i \in [n]$, parameters ε, δ , output an (ε, δ) -approximation of $F_0(\mathcal{D})$. That is, output c such that $\Pr[(1 - \varepsilon) \cdot F_0(\mathcal{D}) \leq c \leq (1 + \varepsilon) \cdot F_0(\mathcal{D})] \geq 1 - \delta$.*

We are interested in streaming algorithms that uses $\text{poly}(\log m, \log n, \varepsilon^{-1}, \log \delta^{-1})$ bits of memory. Since \mathcal{D} is clear from the context, we also use F_0 to refer to $F_0(\mathcal{D})$.

F_0 estimation problem is a fundamental problem with a long history of theoretical and practical investigations. The seminal work of Flajolet and Martin [9] provided the first algorithm assuming the existence of hash functions with full independence. Subsequent

¹ The authors decided to forgo the old convention of alphabetical ordering of authors in favor of a randomized ordering, denoted by . The publicly verifiable record of the randomization is available at <https://www.aeaweb.org/journals/policies/random-author-order/search>



investigations relying on the usage of limited-independence hash functions have led to design of algorithms with optimal space complexity $O(\log n + \frac{\log \delta^{-1}}{\varepsilon^2})$. We defer detailed bibliographical remarks to Section 3. However, all the current space-efficient algorithms are beyond the reach of an undergraduate textbook due to their reliance on notions such as pairwise independence and universal hash functions.

We present a very simple algorithm for the F_0 estimation problem using a sampling strategy that only relies on basic probability for its analysis. In particular, it does not use universal hash functions. While the simplicity of the code makes it appealing to be used in practical implementation, we believe that only using basic probability theory for the analysis makes the algorithm presentable to undergraduates right after the introduction of basic tail bounds. Our algorithm builds and refines ideas introduced in the recent work on estimating the size of the union of sets in the general setting of *Delphic sets* [13].

2 F_0 -Estimator: A simple algorithm for F_0 estimation

Algorithm 1 F_0 -Estimator.

Input Stream $\mathcal{D} = \langle a_1, a_2, \dots, a_m \rangle$, ε, δ

- 1: **Initialize** $p \leftarrow 1$; $\mathcal{X} \leftarrow \emptyset$; $\text{thresh} \leftarrow \frac{12}{\varepsilon^2} \log(\frac{8m}{\delta})$
- 2: **for** $i = 1$ to m **do**
- 3: $\mathcal{X} \leftarrow \mathcal{X} \setminus \{a_i\}$
- 4: With probability p , $\mathcal{X} \leftarrow \mathcal{X} \cup \{a_i\}$
- 5: **if** $|\mathcal{X}| = \text{thresh}$ **then**
- 6: Throw away each element of \mathcal{X} with probability $\frac{1}{2}$
- 7: $p \leftarrow \frac{p}{2}$
- 8: **if** $|\mathcal{X}| = \text{thresh}$ **then**
- 9: **Output** \perp
- 10: **Output** $\frac{|\mathcal{X}|}{p}$

The algorithm F_0 -Estimator uses a simple sampling strategy: it keeps a set \mathcal{X} of samples at all times such that every element seen so far is independently in \mathcal{X} with equal probability. In order to keep the set of samples small, it makes sure that \mathcal{X} does not grow beyond the value thresh by adjusting the sampling rate p accordingly. After all the elements of the stream are processed, it outputs $\frac{|\mathcal{X}|}{p}$ where p is the final sampling rate.

2.1 Theoretical Analysis

We present the theoretical analysis entirely based on first principles, which adds to its length. For readers who are familiar with randomized algorithms, the proof is standard.

We state the following well-known concentration bound, Chernoff bound, for completeness.

► **Fact 2 (Chernoff's Bound).** *Let v_1, \dots, v_k be independent random variables taking values in $\{0, 1\}$. Let $V = \sum_{i=1}^k v_i$ and $\mu = \mathbb{E}[V]$. Then $\Pr(|V - \mu| \geq \delta\mu) \leq 2e^{-\frac{\delta^2\mu}{3}}$*

The following theorem captures the correctness and space complexity guarantee of F_0 -Estimator.

► **Theorem 3.** *For any data stream \mathcal{D} and any $0 < \delta, \varepsilon < 1$, the algorithm F_0 -Estimator outputs an (ε, δ) -approximation of $F_0(\mathcal{D})$. The algorithm uses $O(\frac{1}{\varepsilon^2} \cdot \log n \cdot (\log m + \log 1/\delta))$ space in the worst case.*

Proof. The stated space complexity bound of the algorithm follows because, from the description, it is clear that the size of the set of samples kept by the algorithm is always $\leq \text{thresh}$, and each item requires $\lceil \log_2 n \rceil$ bits to store.

In order to prove the algorithm outputs the correct estimate with high probability, we show that when the algorithm terminates, every distinct element of stream \mathcal{D} is in \mathcal{X} with a probability p , where $p \geq \frac{\text{thresh}}{4F_0}$. This guarantee, together with the Chernoff bound, implies the correctness of the algorithm. We give a formal proof of correctness below.

Consider the following two events:

Error : ‘The algorithm F_0 -Estimator does not return a value in the range $[(1 - \varepsilon)F_0, (1 + \varepsilon)F_0]$ ’
Fail : ‘The algorithm F_0 -Estimator outputs \perp .’

We will bound $\Pr[\text{Error}]$ by δ . Observe that $\Pr[\text{Error}] \leq \Pr[\text{Fail}] + \Pr[\text{Error} \cap \overline{\text{Fail}}]$.

▷ **Claim 4.** $\Pr[\text{Fail}] \leq \frac{\delta}{8}$

Proof of Claim. Let Fail_j denote the event that Algorithm 1 returns \perp when $i = j$. Formally, Fail_j : ‘ $|\mathcal{X}| = \text{thresh}$ and none of the elements of \mathcal{X} are thrown away at line 6’ for $i = j$. The probability that Fail_j happens is $(\frac{1}{2})^{\text{thresh}}$. Therefore,

$$\Pr[\text{Fail}] \leq \sum_{j=1}^m \Pr[\text{Fail}_j] \leq m \cdot \left(\frac{1}{2}\right)^{\text{thresh}} \leq \frac{\delta}{8} \quad \triangleleft$$

▷ **Claim 5.** $\Pr[\text{Error} \cap \overline{\text{Fail}}] \leq \frac{\delta}{2}$.

We give a detailed proof of this claim below. Theorem follows from the above two claims. ◀

Proof of Claim 5

To bound $\Pr[\text{Error} \cap \overline{\text{Fail}}]$, we consider a relaxed version of Algorithm F_0 -Estimator, which is stated as Algorithm 2. Algorithm 2 is nothing but F_0 -Estimator with lines 8 and 9 removed. Observe that for a given input, the algorithm F_0 -Estimator behaves identically to Algorithm 2 as long as $|\mathcal{X}| \leq \text{thresh}$ after each element of \mathcal{X} is thrown away with probability $\frac{1}{2}$ (i.e., the event **Fail** does not happen). Now, we consider the following event:

Error₂ : ‘The Algorithm 2 does not output a value in the range $[(1 - \varepsilon)F_0, (1 + \varepsilon)F_0]$ ’

Observe that $\Pr[\text{Error} \cap \overline{\text{Fail}}] \leq \Pr[\text{Error}_2]$. In Claim 7, we obtain the desired bound on $\Pr[\text{Error}_2]$ and hence on $\Pr[\text{Error} \cap \overline{\text{Fail}}]$.

To prove an upper bound on $\Pr[\text{Error}_2]$ in Claim 7, we will need the following claim. In the following, we use S_j to denote $\{a_1, a_2, \dots, a_j\}$ – distinct elements that appear in the first j items in the stream.

▷ **Claim 6.** The following loop invariant holds in the **for** loop (lines 2– 7) of Algorithm 2:

Every element in S_j is in \mathcal{X} independently with probability p .

Proof. First, we show that if the loop invariant holds after execution of line 4, then it holds after the execution of if-then-block (line 5–7). Since every element of \mathcal{X} is thrown away independently with probability $\frac{1}{2}$ and p is updated to $p/2$, the invariant holds after the execution of the if-then-block (line 5–7).

Now we return our attention to proving that the invariant holds after the execution of line 4 for every iteration j . The proof proceeds via induction.

Algorithm 2

Input Stream $\mathcal{D} = \langle a_1, a_2, \dots, a_m \rangle$, ε , δ

- 1: **Initialize** $p \leftarrow 1$; $\mathcal{X} \leftarrow \emptyset$; $\text{thresh} \leftarrow \frac{12}{\varepsilon^2} \log\left(\frac{8m}{\delta}\right)$
- 2: **for** $i = 1$ to m **do**
- 3: $\mathcal{X} \leftarrow \mathcal{X} \setminus \{a_i\}$
- 4: With probability p , $\mathcal{X} \leftarrow \mathcal{X} \cup \{a_i\}$
- 5: **if** $|\mathcal{X}| = \text{thresh}$ **then**
- 6: Throw away each element of \mathcal{X} with probability $\frac{1}{2}$
- 7: $p \leftarrow \frac{p}{2}$
- 8: **Output** $\frac{|\mathcal{X}|}{p}$

Base Case. After line 4, $\Pr[a_1 \in \mathcal{X}] = p$. Since $\text{thresh} > 1$, the condition in line 5 is not satisfied, therefore the desired invariant holds true.

Inductive Step. Let us assume by induction hypothesis, the desired invariant holds true after iteration $j - 1$. Note that the execution of line 3–line 4 only affects whether $a_j \in \mathcal{X}$ independently of all $a_k \in S_j \setminus a_j$. There are two cases:

- $a_j \notin S_{j-1}$, then after the execution of line 4, we have $\Pr[a_j \in \mathcal{X}] = p$.
- $a_j \in S_{j-1}$, then after line 3, we have $\Pr[a_j \in \mathcal{X}] = 0$. After line 4, we have $\Pr[a_j \in \mathcal{X}] = p$. \triangleleft

▷ **Claim 7.** $\Pr[\text{Error}_2] \leq \frac{\delta}{2}$

Proof. Given the loop invariant stated in Claim 6, we have that every element of S_m is in \mathcal{X} with probability p . In order to upper bound that the probability that $\frac{|\mathcal{X}|}{p}$ lies outside $[(1 - \varepsilon)F_0, (1 + \varepsilon)F_0]$ we will use Chernoff's bound. For this we first establish a lower bound on p . To this end, we decompose $\Pr[\text{Error}_2]$ conditioned on lower bound on p . In particular, we define the following event:

Bad: “The value of p at line 8 in Algorithm 2 is less than $\frac{\text{thresh}}{4F_0}$.”

Let $\ell = \lceil \log\left(\frac{\text{thresh}}{4F_0}\right) \rceil$. Since every value of p can be expressed as power of 2, we have that $p < 2^\ell$ if and only if $p < \frac{\text{thresh}}{4F_0}$. Observe that $\Pr[\text{Error}_2] \leq \Pr[\text{Bad}] + \Pr[\text{Error}_2 \mid \overline{\text{Bad}}]$. We will upper bound $\Pr[\text{Bad}]$ and $\Pr[\text{Error}_2 \mid \overline{\text{Bad}}]$ separately.

Bounding $\Pr[\text{Bad}]$. For $j \in [1, m]$, let Bad_j denote the event that ‘ j th iteration of the **for** loop is the first iteration where the value of p goes below 2^ℓ ’ i.e., the value of p at the end of iteration $(j - 1)$ is 2^ℓ and the value of p is $2^{\ell-1}$ at the end of iteration j . Therefore, by definition of Bad_j , we have $|\mathcal{X}| = \text{thresh}$ and $p = 2^\ell$ in line 5 of Algorithm 2. Recall that in every iteration of the loop, the value of p can decrease at most by a factor of $\frac{1}{2}$ and cannot increase. Therefore, we have $\Pr[\text{Bad}] \leq \sum_{j=1}^m \Pr[\text{Bad}_j]$. We will now compute $\Pr[\text{Bad}_j]$ for a fixed j .

For $a \in S_m$ let r_a denote the indicator random variable indicating whether a is in the set \mathcal{X} . By Claim 6 the random variables $\{r_a\}_{a \in S_m}$ are independent and for all $a \in S_m$ $\Pr[r_a = 1] = p$. Since, $|\mathcal{X}| = \sum_{a \in S_j} r_a$ we have $\mathbb{E}[|\mathcal{X}|] = \mathbb{E}\left[\sum_{a \in S_j} r_a\right] = \sum_{a \in S_j} \Pr[r_a = 1] = p \cdot |S_j| = p \cdot F_0$. Thus,

$$\begin{aligned} \Pr[\text{Bad}_j] &\leq \Pr[|\mathcal{X}| = \text{thresh} \mid p = 2^\ell] \leq \Pr[|\mathcal{X}| \geq \text{thresh} \mid p = 2^\ell] \leq 2e^{-\frac{\text{thresh}^2}{3 \cdot p F_0}} \\ &= 2e^{-\frac{\text{thresh}^2}{3 \cdot 2^\ell F_0}} \leq \frac{\delta}{4m}, \end{aligned}$$

the last inequality follows from the values of ℓ and thresh . Therefore, $\Pr[\text{Bad}] \leq \frac{\delta}{4}$.

Bounding $\Pr[\text{Error}_2 \mid \overline{\text{Bad}}]$. Similar to the above, for $a \in S_m$, let r_a denote the indicator random variable indicating whether a is in the set \mathcal{X} . By Claim 6 the random variables $\{r_a\}_{a \in S_m}$ are independent and for all $a \in S_m$, we have $\Pr[r_a = 1] = p$. Thus $|\mathcal{X}| = \sum_{a \in \mathcal{D}} r_a$ and thus $\mathbb{E}[|\mathcal{X}|] = pF_0$. Conditioned on $\overline{\text{Bad}}$, we have $p \geq \frac{\text{thresh}}{4F_0}$. Thus,

$$\begin{aligned} \Pr[\text{Error}_2 \mid \overline{\text{Bad}}] &= \Pr[(1 - \epsilon)pF_0 \leq |\mathcal{X}| \leq (1 + \epsilon)pF_0 \mid \overline{\text{Bad}}] \\ &\leq 2e^{-\frac{\epsilon^2 \text{thresh}}{12}} \quad \left[\text{Using Chernoff bound with } p \geq \frac{\text{thresh}}{4F_0} \right] \\ &\leq \frac{\delta}{4m} \leq \frac{\delta}{4}. \end{aligned} \quad \triangleleft$$

3 Bibliographic Remarks

Distinct Elements problem (or F_0 estimation problem) is one of the most investigated problem in the data streaming model [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. While the Distinct Elements problem has a wide range of applications in several areas of computing, it was first investigated in the algorithms community by Flajolet and Martin [9]. They provided the first approximation under the assumption of the existence of hash functions with full independence. The seminal work of Alon, Matias, and Szegedy [1] that introduced the data streaming model of computation revisited this problem as a special case of F_k estimation problem and achieved space complexity of $O(\log n)$ for $\epsilon > 1$ and constant δ . The first (ϵ, δ) approximation for Distinct Elements problem was Gibbson and Tirthpura who achieved $O(\frac{\log n}{\epsilon^2})$ space complexity [10]. Bar-Yossef, Jayram, Kumar, Sivakumar and Trevisan improved the space complexity bound to $\tilde{O}(\log n + 1/\epsilon^2)$ [2]. Subsequently, Kane, Nelson, and Woodruff achieved $O(\log n + 1/\epsilon^2)$ which is optimal in n and ϵ [12]. All the above bounds are for a fixed confidence parameter δ , which can be amplified to achieve confidence bounds for arbitrary δ by simply running $\log(1/\delta)$ -estimators in parallel and returning the median. This incurs a multiplicative factor of $\log(1/\delta)$. Błasiok designed an (ϵ, δ) approximation algorithm for F_0 estimation problem with space complexity of $O(\frac{\log \delta^{-1}}{\epsilon^2} + \log n)$, thereby matching the lower bound in all the three parameters n, ϵ and δ [4]. As is expected, every subsequent improvement added to the complexity of the algorithm or the analysis, and a majority of these work remain beyond the reach of non-experts. A crucial technical ingredient for all the works mentioned above is their careful usage of limited-independence hash functions in order to make space $\text{poly}(\log n)$. Monte Carlo-based approaches have been utilized in the context of size estimation of the union of sets, but their straightforward adaptation to the streaming setting did not seem to yield progress. Recently, a new sampling-based approach was proposed in the context of estimating the size of the union of sets in the streaming model that achieves space complexity with $\log m$ -dependence [13]. The algorithm we presented adapts ideas from this work to the context of F_0 estimation.

References


- 1 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. doi:10.1006/jcss.1997.1545.
- 2 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proc. of RANDOM*, pages 1–10, 2002. doi:10.1007/3-540-45726-7_1.

- 3 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002. URL: <http://dl.acm.org/citation.cfm?id=545381.545464>.
- 4 Jaroslaw Blasiok. Optimal streaming and tracking distinct elements with high probability. In *Proc. of SODA*, 2018. doi:10.1137/1.9781611975031.156.
- 5 Joshua Brody and Amit Chakrabarti. A multi-round communication lower bound for gap hamming and some consequences. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009*, pages 358–368. IEEE Computer Society, 2009. doi:10.1109/CCC.2009.31.
- 6 Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In *Algorithms – ESA 2003, 11th Annual European Symposium*, volume 2832, pages 605–617, 2003. doi:10.1007/978-3-540-39658-1_55.
- 7 Cristian Estan, George Varghese, and Michael E. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, 2006. doi:10.1145/1217709.
- 8 Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156, 2007. doi:10.46298/dmtcs.3545.
- 9 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985. doi:10.1016/0022-0000(85)90041-8.
- 10 Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA*, pages 281–291, 2001. doi:10.1145/378580.378687.
- 11 Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem. In *Proc. of FOCS*, pages 283–288. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238202.
- 12 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 41–52, 2010. doi:10.1145/1807085.1807094.
- 13 Kuldeep S. Meel, N. V. Vinodchandran, and Sourav Chakraborty. Estimating the size of union of sets in streaming models. In *PODS’21: Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 126–137, 2021. doi:10.1145/3452021.3458333.

Approximate Circular Pattern Matching

Panagiotis Charalampopoulos  

Birkbeck, University of London, UK
Reichman University, Herzliya, Israel

Tomasz Kociumaka  

Max Planck Institute for Informatics, Saarbrücken, Germany

Jakub Radoszewski  

University of Warsaw, Poland
Samsung R&D, Warsaw, Poland

Solon P. Pissis  

CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

Wojciech Rytter  

University of Warsaw, Poland

Tomasz Waleń  

University of Warsaw, Poland

Wiktor Zuba  

CWI, Amsterdam, The Netherlands

Abstract

We investigate the complexity of approximate *circular pattern matching* (CPM, in short) under the *Hamming* and *edit* distance. Under each of these two basic metrics, we are given a length- n text T , a length- m pattern P , and a positive integer threshold k , and we are to report all starting positions (called occurrences) of fragments of T that are at distance at most k from some cyclic rotation of P . In the decision version of the problem, we are to check if there is any such occurrence. All previous results for approximate CPM were either average-case upper bounds or heuristics, with the exception of the work of Charalampopoulos et al. [CKP⁺, JCSS'21], who considered only the Hamming distance. For the reporting version of the approximate CPM problem, under the Hamming distance we improve upon the main algorithm of [CKP⁺, JCSS'21] from $\mathcal{O}(n + (n/m) \cdot k^4)$ to $\mathcal{O}(n + (n/m) \cdot k^3 \log \log k)$ time; for the edit distance, we give an $\mathcal{O}(nk^2)$ -time algorithm. Notably, for the decision versions and wide parameter-ranges, we give algorithms whose complexities are almost identical to the state-of-the-art for standard (i.e., non-circular) approximate pattern matching:

- For the decision version of the approximate CPM problem under the Hamming distance, we obtain an $\mathcal{O}(n + (n/m) \cdot k^2 \log k / \log \log k)$ -time algorithm, which works in $\mathcal{O}(n)$ time whenever $k = \mathcal{O}(\sqrt{m \log \log m / \log m})$. In comparison, the fastest algorithm for the standard counterpart of the problem, by Chan et al. [CGKKP, STOC'20], runs in $\mathcal{O}(n)$ time only for $k = \mathcal{O}(\sqrt{m})$. We achieve this result via a reduction to a geometric problem by building on ideas from [CKP⁺, JCSS'21] and Charalampopoulos et al. [CKW, FOCS'20].
- For the decision version of the approximate CPM problem under the edit distance, the $\mathcal{O}(nk \log^3 k)$ runtime of our algorithm near matches the $\mathcal{O}(nk)$ runtime of the Landau–Vishkin algorithm [LV, J. Algorithms'89] for approximate pattern matching under edit distance; the latter algorithm remains the fastest known for $k = \Omega(m^{2/5})$. As a stepping stone, we propose an $\mathcal{O}(nk \log^3 k)$ -time algorithm for solving the Longest Prefix k' -Approximate Match problem, proposed by Landau et al. [LMS, SICOMP'98], for all $k' \in \{1, \dots, k\}$. Our algorithm is based on Tiskin's theory of *seaweeds* [Tiskin, Math. Comput. Sci.'08], with recent advancements (see Charalampopoulos et al. [CKW, FOCS'22]), and on exploiting the seaweeds' relation to *Monge matrices*.

In contrast, we obtain a conditional lower bound that suggests a polynomial separation between approximate CPM under the Hamming distance over the binary alphabet and its non-circular counterpart. We also show that a strongly subquadratic-time algorithm for the decision version of approximate CPM under edit distance would refute the Strong Exponential Time Hypothesis.



© Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 35; pp. 35:1–35:19



Leibniz International Proceedings in Informatics

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

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases approximate circular pattern matching, Hamming distance, edit distance

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.35

Related Version *Full Version*: <http://arxiv.org/abs/2208.08915>

Funding *Panagiotis Charalampopoulos*: Supported by Israel Science Foundation (ISF) grant 810/21 when the work that led to this paper was conducted.

Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

Solon P. Pissis: Supported by the PANGAIA and ALPACA projects that have received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 872539 and 956229, respectively.

Tomasz Walęń: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

Wiktor Zuba: Supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.

Acknowledgements We thank Paweł Gawrychowski and Oren Weimann for suggesting the strategy for the proof of Lemma 29.

1 Introduction

Pattern matching is one of the most widely-studied problems in computer science. Given a string P of length m , known as the *pattern*, and a string T of length $n \geq m$, known as the *text*, the task is to compute all occurrences of P in T . In the standard setting, the matching relation between P and the fragments of T assumes that the leftmost and rightmost positions of the pattern are conceptually important. In many real-world applications, however, any rotation (cyclic shift) of P is a relevant pattern. For instance, in bioinformatics [3, 6, 27, 33], the position where a sequence starts can be totally arbitrary due to arbitrariness in the sequencing of a circular molecular structure or due to inconsistencies introduced into sequence databases as a result of different linearisation standards [3]. In image processing [2, 40, 41, 42], the contours of a shape may be represented through a directional chain code; the latter can be interpreted as a cyclic sequence if the orientation of the image is not important [2].

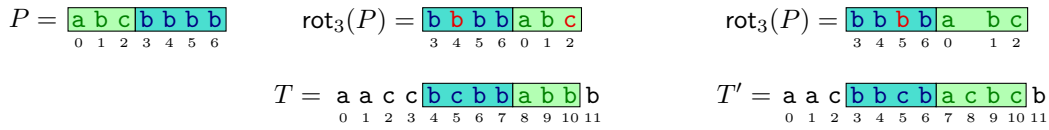
With such scenarios in mind, when matching a pattern P against a text T , one is interested in computing all fragments of T that match some rotation of P . Let us introduce necessary basic notation. The positions of a string U are numbered from 0 to $|U| - 1$, with $U[i]$ denoting the i -th letter, and $U[i..j] = U[i..j+1)$ denoting the substring $U[i] \cdots U[j]$, which is empty if $i > j$. We now formally define the circular pattern matching problem.

CIRCULAR PATTERN MATCHING (CPM)

Input: A text T of length n and a pattern P of length m .

Output: The set $\{i : T[i..i+m) = P' \text{ for some rotation } P' \text{ of } P\}$.

A textbook solution for CPM works in $\mathcal{O}(n \log \sigma)$ time (or $\mathcal{O}(n)$ time with randomization), where σ is the alphabet’s size, using the suffix automaton of $P \cdot P$ [39]. There is a simple deterministic $\mathcal{O}(n)$ -time algorithm which we discuss in the full version. Many practically fast algorithms for CPM also exist; see [17, 23, 43] and references therein. For the indexing version of the CPM problem (searching a collection of circular patterns), see [31, 32, 33].



■ **Figure 1** For a pattern $P = UV$, with $|U| = x$, we denote the rotation VU of P by $\text{rot}_x(P)$. Middle: a circular 2-mismatch occurrence of pattern P (left) at position 4 in text T . Right: a circular 2-edit occurrence of pattern P with the same rotation at position 3 in text T' (note that there is no circular 2-mismatch occurrence of P in T' at this position).

As in the standard pattern matching setting, a single surplus or missing letter in P or in T may result in many occurrences being missed. In bioinformatics, this may correspond to a single-nucleotide polymorphism; in image processing, this may correspond to data corruption. Thus, a relatively large body of works has been devoted to practically fast algorithms for *approximate* CPM; see [2, 4, 5, 7, 8, 24, 29, 30] and references therein. All previous results for approximate CPM were either average-case upper bounds or heuristics, with the exception of the work of Charalampopoulos et al. [14].

Here, like in the previous works on approximate CPM, we consider two well-known approximate matching relations of two strings U and V : the Hamming distance, denoted as $\delta_H(U, V)$ (the number of mismatches for two equal-length strings, otherwise equal to ∞), and the edit distance $\delta_E(U, V)$ (the minimum number of insertions, deletions and substitutions required to transform U to V). For two strings U and V , an integer $k > 0$, and a distance function d on strings, we write $U \stackrel{d}{=} V$ if $d(U, V) \leq k$ and $U \stackrel{d}{\approx} V$ if there exists a rotation U' of U such that $U' \stackrel{d}{=} V$. We define $\text{CircOcc}_k^d(P, T) = \{i : P \stackrel{d}{\approx} T[i..p] \text{ for some } p \geq i\}$; we call it the set of *circular k -mismatch (k -edit) occurrences of P in T* if $d = \delta_H$ ($d = \delta_E$, respectively). We omit the d -superscript when it is clear from the context. We next formally define four variants of k -approximate CPM; see Figure 1 for an example.

k -APPROXIMATE CPM: k -MISMATCH CPM AND k -EDIT CPM

Input: A text T of length n , a pattern P of length m , a positive integer k , and a distance function d : $d = \delta_H$ for k -Mismatch CPM and $d = \delta_E$ for k -Edit CPM.

Output: (**Reporting**) $\text{CircOcc}_k^d(P, T)$.
 (**Decision**) Any position $i \in \text{CircOcc}_k^d(P, T)$, if it exists.

Our upper bounds for k -Mismatch CPM. A summary of the best previous and our new worst-case upper bounds on approximate CPM for strings over a polynomially-bounded integer alphabet (we omit “polynomially-bounded” henceforth) is provided in Table 1.

■ **Table 1** Comparison of previous upper bounds and our results on k -approximate CPM.

Distance Metric	Time Complexity	Version	Reference
Hamming distance	$\mathcal{O}(nk)$	reporting	[14]
	$\mathcal{O}(n + (n/m) \cdot k^4)$		
	$\mathcal{O}(n + (n/m) \cdot k^3 \log \log k)$	reporting	this work
	$\mathcal{O}(n + (n/m) \cdot k^2 \log k / \log \log k)$	decision	
Edit distance	$\mathcal{O}(nk^2)$	reporting	this work
	$\mathcal{O}(nk \log^3 k)$	decision	

35:4 Approximate Circular Pattern Matching

In Section 2 we prove the following results.

► **Theorem 1.** *The reporting version of k -Mismatch CPM for strings over an integer alphabet can be solved in $\mathcal{O}(n + (n/m) \cdot k^3 \log \log k + \text{Output})$ time.*

► **Theorem 2.** *The decision version of k -Mismatch CPM for strings over an integer alphabet can be solved in $\mathcal{O}(n + (n/m) \cdot k^2 \log k / \log \log k)$ time.*

Our upper bounds for k -Edit CPM. A proof of the following theorem, based on the classic Landau–Vishkin algorithm [38], is given in Section 3.

► **Theorem 3.** *The reporting version of k -Edit CPM for strings over an integer alphabet can be solved in $\mathcal{O}(nk^2)$ time.*

We reduce the decision version of k -Edit CPM to the following problem.

LONGEST PREFIX k -APPROXIMATE MATCH (k -LPAM)

Input: A text T of length n and a pattern P .

Output: An array $\text{LPref}_k[0..n]$ such that $\text{LPref}_k[i]$ is the length of the longest prefix of P that matches a prefix of $T[i..n)$ with at most k edits.

Specifically, we introduce a problem called *All- k -LPAM* that consists in solving k' -LPAM for all $k' \in [0..k]$. We show that k -Edit CPM can be reduced to All- k -LPAM on the same pattern and text and on the reversed pattern and text. Landau et al. [37] gave an $\mathcal{O}(nk)$ -time solution to k -LPAM, which provides an $\mathcal{O}(nk^2)$ -time solution to All- k -LPAM. In Section 4 we show the following result for All- k -LPAM (Theorem 4) which implies Theorem 5; All- k -LPAM can find further applications in approximate pattern matching; for example see [9].

► **Theorem 4.** *All- k -LPAM for strings over an integer alphabet can be solved in $\mathcal{O}(nk \log^3 k)$ time.*

► **Theorem 5.** *The decision version of k -Edit CPM for strings over an integer alphabet can be solved in $\mathcal{O}(nk \log^3 k)$ time.*

The complexities of our algorithms for the decision versions of k -Mismatch and k -Edit CPM match, up to $\log^{\mathcal{O}(1)} k$ factors, the complexities of some of the fastest known algorithms for pattern matching with up to k mismatches [12, 15, 21, 26] and edits [38], respectively.

In [37], an algorithm for a weaker problem of computing, given two strings U and V each of length at most n , the rotation of U with the minimum edit distance to V is given. The algorithm works in $\mathcal{O}(ne)$ time, where e is the minimum edit distance achieved.

Our conditional lower bounds. We reduce known problems to approximate CPM, as shown in Table 2.

■ **Table 2** Our conditional lower bounds for approximate CPM for alphabets of constant size.

Problem	Conditioned on	Complexity	Reference
Mismatch-CPM (unbounded)	BJI	$\Omega(n^{1.5-\varepsilon})$ for all const. $\varepsilon > 0$	Theorem 30
k -Edit CPM (decision)	SETH	$\Omega(n^{2-\varepsilon})$ for all const. $\varepsilon > 0$	Theorem 31

For the Hamming distance, we consider the Mismatch-CPM problem where the number of allowed mismatches is unbounded (see Section 5 for a precise definition). The breakthrough work constructing a Binary Jumbled Index (BJI) in $\mathcal{O}(n^{1.859})$ time [13] was very recently improved to $\mathcal{O}(n^{1.5} \log^{\mathcal{O}(1)} n)$ time [18]. We show that obtaining an $\mathcal{O}(n^{1.5-\varepsilon})$ -time algorithm, for any constant $\varepsilon > 0$, for Mismatch-CPM over the binary alphabet would require a further improvement to BJI. A similar problem of (non-circular) pattern matching with mismatches has a classic $\mathcal{O}(n \log n)$ -time solution for constant-size alphabets using convolutions [22]; and the fastest known solution for a general alphabet works in $\mathcal{O}(n^{1.5} \sqrt{\log n})$ time [1, 36]. Our conditional lower bound for k -Edit CPM is based on the conditional hardness of computing the edit distance of two binary strings [10], which in turn is based on the Strong Exponential Time Hypothesis (SETH) [34]. It implies conditional optimality of our algorithm for the decision version of k -Edits CPM for a general $k \leq n$ up to a subpolynomial factor.

The PILLAR model. For k -Mismatch CPM, we work in the PILLAR model that was introduced in [15] with the aim of unifying approximate pattern matching algorithms across different settings. In this model, we assume that the following primitive PILLAR operations can be performed efficiently, where the argument strings are substrings of strings in a collection \mathcal{X} , represented via handles:

- **Extract**(S, ℓ, r): Retrieve string $S[\ell..r]$.
- **LCP**(S, T), **LCP_R**(S, T): Compute the length of the longest common prefix/suffix of S, T .
- **IPM**(S, T): Assuming that $|T| \leq 2|S|$, compute the starting positions of all exact occurrences of S in T , expressed as an arithmetic sequence.
- **Access**(S, i): Retrieve the letter $S[i]$.
- **Length**(S): Compute the length $|S|$ of the string S .

In fact, in the standard setting, where the strings are given explicitly and are over an integer alphabet, all primitive PILLAR operations can be performed in $\mathcal{O}(1)$ time after a linear-time preprocessing. The runtime of algorithms in this model can be expressed in terms of the number of primitive PILLAR operations. (Any extra time required by our algorithms is explicitly specified.)

In the full version we obtain fast algorithms for k -Mismatch CPM in the internal, dynamic and fully compressed setting, by using Theorems 1 and 2 with known implementations of the primitive PILLAR operations in these settings.

2 k -Mismatch CPM

For a string S and an integer $x \leq |S|$, we denote $\text{rot}_x(S) = S[x..|S|] \cdot S[0..x]$. For $i \in [0..|S| - m]$, we denote $S^{(i)} = S[i..i + m]$. Also, we denote the set of standard (non-circular) k -mismatch occurrences of P in T by $\text{Occ}_k(P, T) = \{i \in [0..n - m] : T^{(i)} =_k P\}$.

Let $P = P_1 P_2$, where $|P_1| = \lfloor m/2 \rfloor$. Each circular k -mismatch occurrence of P in T implies a standard k -mismatch occurrence of at least one of P_1 and P_2 . Henceforth, we assume, without loss of generality, that it implies a k -mismatch occurrence of P_1 , as the remaining case is symmetric. Our goal is to consider all k -mismatch occurrences of P_1 in T as anchors for computing circular k -mismatch occurrences of P in T . We call P_1 *the sample*.

By the so-called standard trick, we will consider $\mathcal{O}(n/m)$ substrings of T of length $\mathcal{O}(m)$ and process each of them separately. We denote one such substring by T' . All positions (occurrences) of the sample can be efficiently computed in such a small *window* T' :

► **Theorem 6** ([15, Main Theorems 5 and 8]). *Given a text T' and a pattern P_1 , satisfying $|T'| = \mathcal{O}(|P_1|)$, we can compute a representation of the set $\text{Occ}_k(P_1, T')$ as $\mathcal{O}(k^2)$ pairwise disjoint arithmetic progressions in $\mathcal{O}(k^2 \log \log k)$ time plus $\mathcal{O}(k^2)$ PILLAR operations.¹*

We want to find in T extensions of occurrences of P_1 in T' , which approximately match some rotation of P . Consider only rotations of P of the form YP_1X , where $P = P_1XY$. Define the set of circular k -mismatch occurrences i of P in T that imply a k -mismatch standard occurrence of P_1 in a substring $T' = T[a..b]$ as follows (such occurrences are *anchored* in (P_1, T')):

$$\text{Anchored}_k(P, T, P_1, T') = \{i : T^{(i)} =_k YP_1X, P = P_1XY, i + |Y| - a \in \text{Occ}_k(P_1, T')\}.$$

Our algorithms compute a superset of $\text{Anchored}_k(P, T, P_1, T')$ that may also contain other circular k -mismatch occurrences of P in T (some of the ones that contain a k -mismatch occurrence of P_2). Let $A = \text{Occ}_k(P_1, T')$, and recall that this refers to standard k -mismatch occurrences.

2.1 Few k -mismatch Occurrences of the Sample: $|A| = \mathcal{O}(k)$

We assume that $|A| = \mathcal{O}(k)$. Let us denote by $\text{PAIRMATCH}_k(T, P, i, j)$ the set of all circular k -mismatch occurrences of P in T such that position i in T is aligned with position j in P :

$$\text{PAIRMATCH}_k(T, P, i, j) = \{p \in [i - m + 1 .. i] : T^{(p)} =_k \text{rot}_x(P), i - p \equiv j - x \pmod{m}\}.$$

In particular $\text{PAIRMATCH}_k(T, P, i, 0)$ is the set of circular k -mismatch occurrences of P such that the first position of P is aligned with position i in T .

The following lemma was shown without explicitly mentioning the PILLAR model.

► **Lemma 7** ([14, see Lemma 10]). *$\text{PAIRMATCH}_k(T, P, i, j)$ can be computed in $\mathcal{O}(k)$ time in the PILLAR model for any given k, i, j , with the output represented as $\mathcal{O}(k)$ intervals.*

► **Lemma 8.** *Given $A = \text{Occ}_k(P_1, T')$ of size $\mathcal{O}(k)$, a superset of $\text{Anchored}_k(P, T, P_1, T')$, represented as $\mathcal{O}(k^2)$ intervals, can be computed in $\mathcal{O}(k^2)$ time in the PILLAR model. The computed superset contains only circular k -mismatch occurrences of P in T .*

Proof. Suppose that $T' = T[a..b]$. We then have

$$\text{Anchored}_k(P, T, P_1, T') \subseteq \bigcup_{i \in A} \text{PAIRMATCH}_k(T, P, i + a, 0) \subseteq \text{CircOcc}_k(P, T).$$

By the hypothesis $|A| = \mathcal{O}(k)$, the result, represented as a union of $\mathcal{O}(k^2)$ intervals, can be computed in $\mathcal{O}(k^2)$ time in the PILLAR model by means of Lemma 7. ◀

2.2 Many k -mismatch Occurrences of the Sample: $|A| > 864k$

We assume that $|A| = |\text{Occ}_k(P_1, T')| > 864k$. By [15], these k -mismatch occurrences imply approximate periodicity in both P_1 and the portion of T' spanned by k -mismatch occurrences of P_1 . We show that, except for $\mathcal{O}(k^2)$ intervals of circular k -mismatch occurrences of P that we can compute using $\mathcal{O}(k)$ calls to PAIRMATCH_k as in Section 2.1, our problem reduces to matching length- m substrings of an approximately periodic substring V in T and of an

¹ When referring to statements of [15], we use their numbering in the full (arxiv) version of the paper.

approximately periodic substring U in P^2 with aligned approximate periodicity. Afterwards, testing a match of two length- m substrings for U and V reduces to the test only on positions breaking periodicity, called *misperiods*, since the equality on other positions is guaranteed due to common approximate periodicities. The number of misperiods in U and V is only $\mathcal{O}(k)$, so the complexity of our algorithms, in terms of PILLAR operations, depends only on k .

By S^p we denote the concatenation of p copies of a string S . A string Q is called primitive if $Q = B^p$ for a string B and a positive integer p implies that $p = 1$. We introduce the following problem.

PERIODICSUBSTRINGMATCH(U, V)

Input: Positive integers m, k , and q , an integer $r \in [0..q)$, and strings U, V , and Q such that $m \leq |U|, |V| \leq 2m$, $q = |Q|$, and U, V are at Hamming distance $\mathcal{O}(k)$ from prefixes of $Q^\infty, (\text{rot}_r(Q))^\infty$, respectively.

Output: The set of positions p in U for which there exists a position x in V such that $U^{(p)} =_k V^{(x)}$ and $p - x \equiv r \pmod{q}$.

Our goal is to reduce k -Mismatch CPM in this case to PERIODICSUBSTRINGMATCH. To this end, we use the idea of repetitive regions from [15].

► **Definition 9.** A k -repetitive region in a string S of length m is a substring R of S of length $|R| \geq 3m/8$ for which there exists a primitive string Q such that

$$|Q| \leq m/(128k) \text{ and } \delta_H(R, Q^\infty[0..|R|]) = \lceil 8k|R|/m \rceil.$$

The following lemma is a simplified version of a lemma from [15] with one repetitive region.

► **Lemma 10** ([15, see Lemma 3.11]). Given a pattern P of length m , a text T of length $n \leq \frac{3}{2}m$, and a positive integer threshold $k \leq m$, if the pattern P contains a k -repetitive region, then $|\text{Occ}_k(P, T)| \leq 864k$.

Intuitively, in our main lemma in this section (Lemma 12), we show that if P_1 has many k -mismatch occurrences in T' , then each circular k -mismatch occurrence of P in T that is an element of $\text{Anchored}_k(P, T, P_1, T')$ is: (a) either computed in an instance of PERIODICSUBSTRINGMATCH; or (b) implies a k -mismatch occurrence of one of at most two k -repetitive regions of $P_2P_1P_2$. The occurrences of the second type can be computed using $\mathcal{O}(k)$ calls to PAIRMATCH $_k$ by viewing k -repetitive regions as samples and applying Lemma 10 to bound the number of k -mismatch occurrences.

In the proof of Lemma 12 we use the following theorem from [15]; part 1 follows from [15, Theorem 3.1] (existence of Q) and [15, Lemmas 3.8, 3.11, 3.14, and 4.4] (computation of Q), whereas the remaining parts are due to [15, Main Theorem 5].

► **Theorem 11** ([15]). Assume that we are given a pattern P of length m , a text T of length $n \leq \frac{3}{2}m$, and a positive integer threshold $k \leq m$. If $|\text{Occ}_k(P, T)| > 864k$ and T starts and ends with k -mismatch occurrences of P , that is, $0, |T| - m \in \text{Occ}_k(P, T)$, then:

1. there is a substring Q of P satisfying $|Q| \leq m/(128k)$ and $\delta_H(P, Q^\infty[0..|P|]) < 2k$, which can be computed in $\mathcal{O}(k)$ time in the PILLAR model;
2. each element of $\text{Occ}_k(P, T)$ is a multiple of $|Q|$;
3. $\delta_H(T, Q^\infty[0..n]) \leq 6k$;
4. $\text{Occ}_k(P, T)$ can be decomposed into $\mathcal{O}(k^2)$ arithmetic progressions with difference $|Q|$.

► **Lemma 12** (Reduction to PERIODICSUBSTRINGMATCH). *If we have $|T'| \leq \lfloor \frac{3}{4}m \rfloor$ and $|\text{Occ}_k(P_1, T')| > 864k$, then there is a superset of $\text{Anchored}_k(P, T, P_1, T')$ containing circular k -mismatch occurrences of P in T that is a union of $\mathcal{O}(k^2)$ intervals and of the output of PERIODICSUBSTRINGMATCH with U and V being substrings of T and P^2 , respectively. The intervals and the input to the PERIODICSUBSTRINGMATCH call can be computed in $\mathcal{O}(k^2 \log \log k)$ time plus $\mathcal{O}(k^2)$ PILLAR operations.*

Proof. We apply Theorem 11 to P_1 and the part T'' of T' spanned by the k -mismatch occurrences in $\text{Occ}_k(P_1, T')$, obtaining a short primitive string Q . Consider the occurrence of P_1 at position $|P_2|$ of $P_2P_1P_2$. Let us extend this substring to the right, trying to accumulate enough mismatches with a prefix of Q^∞ in order to reach the threshold specified in Definition 9, which is $\Theta(k)$. In order to compute the next mismatch, it suffices to perform an LCP query between the remaining suffix of $P_2P_1P_2$ and some rotation of Q^∞ . An analogous process was described in detail in [15, Lemma 4.4], and the fact that the PILLAR model supports the described LCP queries is shown in [15, Corollary 2.9]. If we manage to accumulate enough mismatches, we call the resulting k -repetitive region R_R . We perform the same process by extending this occurrence of P_1 to the left, possibly obtaining a k -repetitive region R_L . Then, we let V be the shortest substring $(P_2P_1P_2)[v..v']$ of $P_2P_1P_2$ that spans both R_L (or P_2P_1 if R_L does not exist) and R_R (or P_1P_2 if R_R does not exist). Let us observe that V is at distance at most $2 \cdot \lceil 8km/m \rceil = 16k$ from a prefix of $(\text{rot}_{r(P)}(Q))^\infty$, where $r(P) = v - |P_2| \pmod{|Q|}$, by the definition of k -repetitive regions and the fact that $|R_R|, |R_L| \leq m$. Moreover, obviously, $|V| \leq 2m$.

The rotations of P that contain P_1 are in one-to-one correspondence with the length- m substrings of $P_2P_1P_2$. Each such substring either contains (at least) one of R_L and R_R or is contained in V . We now show that we can efficiently compute circular k -mismatch occurrences of P that imply k -mismatch occurrences of either R_L or R_R (if they exist) using the tools that developed in Section 2.1. We focus on R_R as R_L can be handled symmetrically. Due to Lemma 10, R_R has $\mathcal{O}(k)$ k -mismatch occurrences in T' , and they can be found in $\mathcal{O}(k^2 \log \log k)$ time plus $\mathcal{O}(k^2)$ PILLAR operations using Theorem 6. For each such occurrence at some position i of T , we perform a call PAIRMATCH_k as in Lemma 8.

We are left with computing elements of $\text{Anchored}_k(P, T, P_1, T')$ corresponding to k -mismatch occurrences of length- m substrings of V in T in the case where $|V| \geq m$. Let us take the considered occurrence $T[a..b]$ of T'' in T , which is at distance at most $6k$ from a prefix of Q^∞ , and extend it to the right until either of the following three conditions is satisfied: (a) we reach the end of T ; (b) we have appended $\lceil m/2 \rceil$ letters; or (c) the resulting substring has $18k$ additional mismatches with the same-length prefix of Q^∞ . We extend the obtained substring to the left until either of the following three conditions is satisfied: (a) we reach the beginning of T ; (b) we have prepended $\lceil m/2 \rceil$ letters; or (c) the prepending substring has $18k$ mismatches with the same-length suffix of $Q^{|T|}$. We set the obtained substring $T[u..u']$ of T to be U . We observe that $|U| \leq 2m$ (since $\lfloor 3m/4 \rfloor \geq |T'| > 864k \Rightarrow m > 1152$ and $|U| \leq |T'| + 2 \lceil m/2 \rceil \leq 7m/4 + 2$) and U is at distance at most $6k + 2 \cdot 18k = 42k$ from a prefix of $(\text{rot}_{r(T)}(Q))^\infty$, where $r(T) = u - a \pmod{|Q|}$. If $|U| < m$, we do not construct the instance of PERIODICSUBSTRINGMATCH.

In the call to PERIODICSUBSTRINGMATCH, we set $Q := \text{rot}_{r(T)}(Q)$ and $r := (r(P) - r(T)) \pmod{|Q|}$. Now, since Q is primitive, it does not match any of its non-trivial rotations. Further, as we have at least $128k - 42k - 1$ (resp. $128k - 16k - 1$) exact occurrences of Q in any length- m fragment of U (resp. V), the periodicities must be synchronized in any circular k -mismatch occurrence. Hence, for any $p \in \text{Anchored}_k(P, T, P_1, T')$ that corresponds to $U^{(p)} =_k V^{(x)}$ we must have $p + r(T) \equiv x + r(P) \pmod{|Q|}$, and hence $p - x \equiv r(P) - r(T) \equiv r \pmod{|Q|}$.

It suffices to show that there is no position $i \in \text{Anchored}_k(P, T, P_1, T')$ such that $T^{(i)}$ is at distance at most k from a substring of V and $[i..i+m] \not\subseteq [u..u']$. Suppose that this is the case towards a contradiction, and assume without loss of generality that $i < u$; the other case is symmetric. We notice that this can only be the case if we stopped extending to the left because we accumulated enough mismatches. Further, let the implied k -mismatch occurrence of P_1 start at some position t of T , let x be an integer such that $\min_y \delta_H(V^{(y)}, T^{(i)}) = \delta_H(V^{(x)}, T^{(i)})$, and let F be the length- $(t-i)$ suffix of $Q^{|T|}$, i.e., $Q^{|T|}[[T] \cdot |Q| - (t-i) .. |T| \cdot |Q|]$. Then, via the triangle inequality, we have

$$\begin{aligned} \delta_H(V^{(x)}, T^{(i)}) &\geq \delta_H(V^{(x)}[0..t-i], T[i..t]) \\ &\geq \delta_H(T[i..t], F) - \delta_H(F, V^{(x)}[0..t-i]) \geq 18k - 16k > k, \end{aligned}$$

thus obtaining a contradiction. This completes the proof of this lemma. \blacktriangleleft

2.3 The Reporting Version of k -Mismatch CPM

In this section, we give a solution to PERIODICSUBSTRINGMATCH. Let us recall the notion of misperiods that was introduced in [14].

► **Definition 13.** A position a in S is a misperiod with respect to a substring Q of S if $S[a] \neq Q^\infty[a]$. We denote the set of misperiods by $\text{Misp}(S, Q)$.

In $\mathcal{O}(k)$ time in the PILLAR model we can compute the sets $I = \text{Misp}(U, Q)$ and $J = \text{Misp}(V, \text{rot}_r(Q))$. This is due to [15, Corollary 2.9], which allows us to answer queries of the form $\text{LCP}(S[i..j], Q^\infty[a..b])$ in $\mathcal{O}(1)$ time in the PILLAR model. For an integer z , let us denote $\mathbf{W}_z = [z..z+m)$ (a window of size m). We define $\text{Mispers}(i, j) = |\mathbf{W}_i \cap I| + |\mathbf{W}_j \cap J|$.

The following problem is a simpler version of PERIODICSUBSTRINGMATCH that was considered in [14]. (Actually, [14] considered a slightly more restricted problem which required that no two misperiods in $U^{(p)}$ and $V^{(x)}$ are aligned and computed a superset of its solution that corresponds exactly to the statement below.)

PERIODICPERIODICMATCH(U, V)

Input: Same as in PERIODICSUBSTRINGMATCH

Output: The set of positions p in U for which there exists a position x in V such that $U^{(p)} =_k V^{(x)}$, $p - x \equiv r \pmod{q}$, and $\text{Mispers}(p, x) \leq k$.

For an integer set A and an integer r , let $A \oplus r = \{a + r : a \in A\}$. An *interval chain* for an interval I and non-negative integers a and q is a set of the form $I \cup (I \oplus q) \cup (I \oplus 2q) \cup \dots \cup (I \oplus aq)$. Here q is called the *difference* of the interval chain.

► **Lemma 14** ([14, Lemma 15]). Given sets $I = \text{Misp}(U, Q)$ and $J = \text{Misp}(V, \text{rot}_r(Q))$, we can compute in $\mathcal{O}(k^2)$ time a solution to PERIODICPERIODICMATCH represented as $\mathcal{O}(k^2)$ interval chains, each with difference q .

Next we observe that if $U^{(p)} =_k V^{(x)}$, then either some two misperiods in $U^{(p)}$ and $V^{(x)}$ are aligned, or the total number of misperiods in these substrings is at most k .

► **Observation 15.** We have $\text{PERIODICSUBSTRINGMATCH}(U, V) =$

$$\text{PERIODICPERIODICMATCH}(U, V) \cup \bigcup_{i \in I, j \in J} \text{PAIRMATCH}_k(U, V, i, j).$$

By the observation, there are $\mathcal{O}(k^2)$ instances of PAIRMATCH_k , each taking $\mathcal{O}(k)$ time in the PILLAR model, and $\text{PERIODICPERIODICMATCH}$ can be solved in $\mathcal{O}(k^2)$ time. This results in total time complexity $\mathcal{O}(k^3)$ for $\text{PERIODICSUBSTRINGMATCH}$. Together with the previous reductions in Lemmas 8 and 12, this leads to Theorem 1. We only need to transform the output from a union of intervals and interval chains to a list of circular k -mismatch occurrences without duplicates. A full proof of Theorem 1 can be found in the full version.

2.4 A Faster Algorithm for the Decision Version of k -Mismatch CPM

Two aligned misperiods can correspond to zero or one mismatch, while two misaligned misperiods always yield two mismatches. Let us recall that $I = \text{Misp}(U, Q)$ and $J = \text{Misp}(V, \text{rot}_r(Q))$. We define the following *mismatch correcting* function:

$$\nabla(i, j) = \begin{cases} 0 & \text{if } (i, j) \notin I \times J, \text{ otherwise:} \\ 1 & \text{if } U[i] \neq V[j], \\ 2 & \text{if } U[i] = V[j]. \end{cases}$$

This function corrects *surplus mismatches*. Let $\text{Surplus}(i, j) = \sum_{t=0}^{m-1} \nabla(i+t, j+t)$.

DECISION $\text{PERIODICSUBSTRINGMATCH}(U, V)$

Input: Same as before, with the sets I, J stored explicitly.

Output: Any position $p \in [0..|U| - m]$ such that $\text{Mispers}(p, x) - \text{Surplus}(p, x) \leq k$ for some $x \in [0..|V| - m]$ such that $p - x \equiv r \pmod{q}$.

We consider an $(|U| - m + 1) \times (|V| - m + 1)$ grid \mathcal{G} . The δ -th diagonal in \mathbb{Z}^2 consists of points (i, j) that satisfy $i - j = \delta$. It is called an *essential* diagonal if $\delta = i - j$ for some $i \in I, j \in J$ and $\delta \equiv r \pmod{q}$. A point (i, j) on an essential diagonal is called *essential* if $i \in I$ and $j \in J$. Let us observe that only essential points influence the function Surplus , and the number of these points is $\mathcal{O}(k^2)$. This implies the following lemma using a simple 1D sweeping algorithm.

► **Lemma 16** (Compact representation of Mispers and Surplus). *In $\mathcal{O}(k^2 \log \log k)$ time we can:*

- (a) *Partition the grid \mathcal{G} by $\mathcal{O}(k)$ vertical and $\mathcal{O}(k)$ horizontal lines into $\mathcal{O}(k^2)$ disjoint rectangles such that for each point (i, j) in a single rectangle the value $\text{Mispers}(i, j)$ is the same. Each rectangle stores the value $\text{Mispers}(i, j)$ for an arbitrary point (i, j) that it contains.*
- (b) *Partition all essential diagonals in \mathcal{G} into $\mathcal{O}(k^2)$ pairwise disjoint diagonal segments, such that for each point (i, j) in a single segment the value $\text{Surplus}(i, j)$ is the same. Each segment stores the value $\text{Surplus}(i, j)$ for an arbitrary point (i, j) that it contains.*

Proof. Partitioning (a): We partition the first axis (second axis) into axis segments such that for all i in the same segment, the set $\mathbf{W}_i \cap I$ ($\mathbf{W}_j \cap J$, respectively) is the same. Then we create rectangles being Cartesian products of the segments.

Partitioning of an axis is performed with a 1D sweep; we describe it in the context of the first axis. For each $i \in I$, we create an event at position $i - m + 1$ where the misperiod i is inserted and an event at position $i + 1$ (if $i + 1 \leq |U| - m + 1$) where it is removed. We can now sort all events in $\mathcal{O}(k \log \log k)$ time using integer sorting [28] and process them in order,

storing the number of misperiods. For all i in a segment without events, the set $\mathbf{W}_i \cap I$ is the same. We obtain $\mathcal{O}(k)$ segments on each axis which yields $\mathcal{O}(k^2)$ rectangles. Part (a) works in $\mathcal{O}(k^2)$ time.

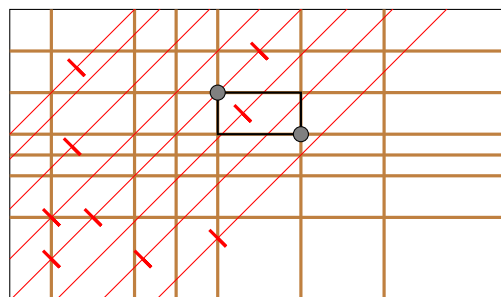
Partitioning (b): First we bucket sort essential points by (essential) diagonals using integer sorting. On each essential diagonal, we sort the essential points bottom-up and perform the same kind of 1D sweep as in (a), using ∇ to compute the weights of the events. The whole algorithm works in $\mathcal{O}(k^2 \log \log k)$ time. ◀

We assume that the grid \mathcal{G} is partitioned by selected horizontal and vertical lines into disjoint rectangles, called *cells*. These cells and some diagonal segments are weighted. Let us denote by $\text{cell}(i, j)$ and $\text{diag}(i, j)$ the weight of the cell and the diagonal segment containing (i, j) , respectively. In the following problem, we care only about points on diagonal segments.

DIAGONALSEGMENTS

Input: A grid partitioned by $\mathcal{O}(k)$ vertical and $\mathcal{O}(k)$ horizontal lines into $\mathcal{O}(k^2)$ weighted rectangles, called cells, and $\mathcal{O}(k^2)$ pairwise disjoint weighted diagonal line segments, all parallel to the line that passes through $(0, 0)$ and $(1, 1)$.

Output: Report a point $(x, y) \in \mathbb{Z}^2$ on some diagonal line segment with minimum $\text{val}(x, y) := \text{cell}(x, y) + \text{diag}(x, y)$.



■ **Figure 2** The weight of the distinguished cell is equal to $\text{Mispers}(i, j)$ for all points (i, j) in that cell. The diagonals are partitioned into segments. The weight of a single diagonal segment is equal to $-\text{Surplus}(i, j)$ for all points (i, j) that lie on that segment. In DIAGONALSEGMENTS we are to find any point (i, j) on some diagonal that minimizes the sum of the weight of its cell and of its diagonal segment, i.e., $\text{Mispers}(i, j) - \text{Surplus}(i, j)$. To this end, it suffices to consider endpoints of diagonal segments and crossings of diagonal segments with rectangles' boundaries.

An intuition of the solution to DIAGONALSEGMENTS is shown in Figure 2.

► **Lemma 17.** *The DIAGONALSEGMENTS problem can be solved in $\mathcal{O}(k^2 \log k / \log \log k)$ time.*

Proof. A sought point (x, y) that minimizes $\text{val}(x, y)$ either (1) is an endpoint of a diagonal segment or (2) lies on an intersection of a diagonal segment and an edge of a cell.

▷ **Claim 18.** Given $\mathcal{O}(p)$ horizontal lines and $\mathcal{O}(p)$ points, one can compute for each point the nearest line above it in $\mathcal{O}(p \log \log p)$ time.

Proof. We create a list of integers containing the vertical coordinates of all queried points and of all lines. Then we sort all of them in $\mathcal{O}(p \log \log p)$ time. The required answers can then be retrieved by a simple traversal of the sorted list. ◀

35:12 Approximate Circular Pattern Matching

In case (1) it suffices to identify, for all end points of all diagonal segments, the cells which they belong to. Let us assume that vertical and horizontal lines partition the grid into columns and rows, respectively. Then each cell can be uniquely identified by its column and row. With Claim 18 we can compute in $\mathcal{O}(k^2 \log \log k)$ time, for all queried points, the rows they belong to; the computation of columns is symmetric.

In case (2), let us consider intersections with horizontal edges; the intersections with vertical edges can be handled symmetrically. Assume x is the horizontal coordinate. We perform an affine transformation of the plane $(x, y) \mapsto (y - x, y)$ after which diagonal line segments become vertical, but horizontal line segments remain horizontal. The sought points can be computed using the following claim in $\mathcal{O}(k^2 \log k / \log \log k)$ time.

▷ **Claim 19.** Given s vertical and horizontal weighted line segments such that no two line segments of the same direction intersect, an intersection point of a vertical and a horizontal line segment with minimum total weight can be computed in $\mathcal{O}(s \log s / \log \log s)$ time.

Proof. We perform a left-to-right line sweep. The events in the sweep are the beginnings and endings of horizontal line segments and vertical line segments. The events can be sorted by their x -coordinates in $\mathcal{O}(s \log \log s)$ time with integer sorting. The horizontal line segments are stored using a dynamic predecessor data structure [46], and their weights in the same order are stored in a dynamic RMQ data structure of size $\mathcal{O}(s)$ that supports insertions, deletions, and range minimum queries in amortized $\mathcal{O}(\log s / \log \log s)$ time [11]. This way, when considering a vertical line segment, we can compute the minimum-weight horizontal line segment that intersects it in $\mathcal{O}(\log s / \log \log s)$ time. ◀

This concludes the solution to DIAGONALSEGMENTS. ◀

► **Theorem 2.** *The decision version of k -Mismatch CPM for strings over an integer alphabet can be solved in $\mathcal{O}(n + (n/m) \cdot k^2 \log k / \log \log k)$ time.*

Proof. Assume that P_1 is the sample. We use the so-called standard trick that covers T by a collection of its substrings T' of length $\lfloor \frac{3}{4}m \rfloor$ starting at positions divisible by $\lfloor \frac{1}{4}m \rfloor$. The following computations are performed for each T' .

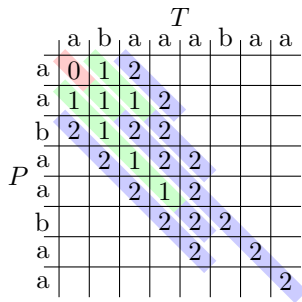
In this paragraph all complexities are stated in the PILLAR model. We use Theorem 6 to compute a representation of the set $A = \text{Occ}(P_1, T')$. If $|A| \leq 864k$, we can use Lemma 8 that outputs $\mathcal{O}(k^2)$ intervals of circular k -mismatch occurrences in $\mathcal{O}(k^2)$ time. Otherwise, we apply Lemma 12 which in $\mathcal{O}(k^2 \log \log k)$ time outputs $\mathcal{O}(k^2)$ intervals of circular k -mismatch occurrences and returns an instance of PERIODICSUBSTRINGMATCH. Next we use the geometric interpretation of PERIODICSUBSTRINGMATCH. The weight of a cell is the value $\text{Mispers}(i, j)$ common to all points (i, j) in this cell. Similarly, the weight of a diagonal segment equals $-\text{Surplus}(i, j)$, common to all points in this segment (it is the number of surplus misperiods which we have to subtract). These values are computed in $\mathcal{O}(k^2 \log \log k)$ time in Lemma 16. Now, the decision version of the PERIODICSUBSTRINGMATCH problem is reduced to the computation of the minimum value of $\text{Mispers}(i, j) - \text{Surplus}(i, j)$, which corresponds to the sum of weights of a cell and diagonal segment meeting at the same point (i, j) . The decision version of PERIODICSUBSTRINGMATCH can be reduced in $\mathcal{O}(k^2 \log \log k)$ time to one instance each of the DIAGONALSEGMENTS (if the sought point is on a diagonal segment) and PERIODICPERIODICMATCH (in the opposite case) problems. The thesis follows from Lemmas 14 and 17. ◀

► **Remark 20.** Using this geometric approach, the reporting version of k -Mismatch CPM can also be solved in $\mathcal{O}(n + (n/m) \cdot k^2 \log^{\mathcal{O}(1)} k + k \cdot \text{Output})$ time.

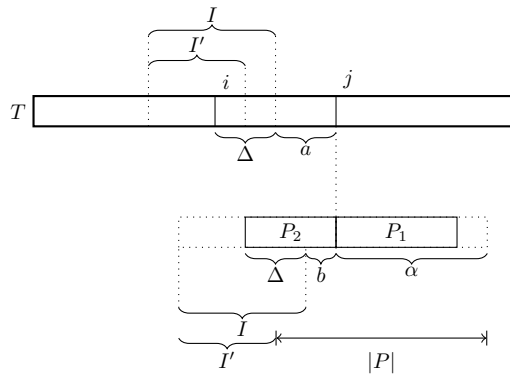
3 *k*-Edit CPM

In the proof of the following lemma, we rely on the Landau–Vishkin algorithm [38]; cf. Figure 3a for an illustration. The details can be found in the full version.

► **Lemma 21.** *Given a text T of length n , a pattern P of length m , and an integer $k > 0$, we can compute in $\mathcal{O}(k^2)$ time in the PILLAR model an $\mathcal{O}(k^2)$ -size representation of the edit distance between all pairs of prefixes of T and P that are at edit distance at most k , that is, the set $LV := \{(a, b, d) \in (0..n] \times (0..m] \times [0..k] : \delta_E(T[0..a], P[0..b]) = d \leq k\}$. Our representation of LV consists of $\mathcal{O}(k^2)$ sets of the form $\{(a + \Delta, b + \Delta, d) : \Delta \in [0..x]\}$.*



(a) Example for Lemma 21:
 $T = abaabaa$, $P = aabaabaa$, $k = 2$.



(b) Notation used in the proof of Lemma 22.

■ **Figure 3** Illustrations for Lemmas 21 and 22.

► **Lemma 22.** *Given a text T of length n , a pattern P of length m , an integer $k > 0$, and a position j of T , after $\mathcal{O}(nk^2)$ preprocessing we can compute in $\mathcal{O}(k^2)$ time in the PILLAR model all positions $i \in \text{CircOcc}_k(P, T) \cap [0..j]$ such that $\delta_E(T[i..j], P_2) + \delta_E(T[j..r], P_1) \leq k$ for some position r of T and some strings P_1 and P_2 that satisfy $P = P_1P_2$, represented as $\mathcal{O}(k^2)$ intervals, possibly with duplicates.*

Proof. We start with the calculation of the compact representation of LV from Lemma 21 for the reversals of strings $T[0..j]$ and P , and k . Next, for each element $\{(a + \Delta, b + \Delta, d) : \Delta \in I\}$ of this compact representation, we will calculate an interval $I' \subseteq I$ such that positions from $\{j - a - \Delta : \Delta \in I'\}$ are in $\text{CircOcc}_k(P, T)$.

From the definition of LV we know that for any $\Delta \in I$, $i = j - a - \Delta$ and $P_2 = P[m - b - \Delta..m]$, we have $\delta_E(T[i..j], P_2) = d$. All we have to do is to verify if there exists P_1 fulfilling requirements of the lemma.

For each $k' \in [0..k]$, we compute $\text{LPref}_{k'}$ in $\mathcal{O}(nk)$ time [37]. Then, the maximal possible length of P_1 (within our edit distance budget) is $\alpha := \text{LPref}_{k-d}[j]$. Now, we need to define I' in such a way that it corresponds to pairs (P_1, P_2) with total length $|P|$, that is, we set $I' := \{\Delta \in I : b + \Delta + \alpha \geq m\}$. The notation used in this proof is illustrated in Figure 3b. The most time consuming part of this procedure is the calculation of the compact representation of LV , which takes $\mathcal{O}(k^2)$ time in the PILLAR model by Lemma 21. ◀

By applying the above lemma for P , T , and all positions j of T , and merging the obtained $\mathcal{O}(nk^2)$ intervals using bucket sort, we obtain Theorem 3.

Let us now move to the decision version of k -Edit CPM. For any integer $k \geq 0$, we define array $\text{LSuf}_k[0..n]$ such that $\text{LSuf}_k[i] = t$ if and only if $P[m-t..m]$ is the longest suffix of P at edit distance at most k from a suffix of $T[0..i]$. We make the following easy observation.

► **Observation 23.** *The pattern P has a circular k -edit occurrence in the text T if and only if $\text{LPref}_{k'}[j] + \text{LSuf}_{k-k'}[j] \geq m$ holds for some $j \in [0..n]$ and $k' \in [0..k]$.*

Due to Observation 23, Theorem 4, which is proved in Section 4, yields Theorem 5.

4 An Algorithm for All- k -LPAM: Proof of Theorem 4

We will show how to compute, given a position p in the text, values $\text{LPref}_{k'}[i]$ for all $k' \in [0..k]$ and $i \in [p.. \min\{p+k, n+1\}]$ in $\mathcal{O}(k^2 \log^3 k)$ time in the PILLAR model. This will yield the desired solution to All- k -LPAM by taking values of p which are multiples of k .

The *deletion distance* $\delta_D(U, V)$ of two strings U and V is the minimum number of letter insertions and deletions required to transform U to V ; in comparison to edit distance, substitutions are not allowed. For a string S , by $S_\$$ we denote the string $S[0]\$S[1]\$\cdots S[|S|-1]\$$. By the following known fact, we can easily transform the pattern and the text, doubling k , and henceforth consider the deletion distance instead of the edit distance.

► **Fact 24.** *For any two strings U and V over an alphabet Σ that does not contain $\$$, we have $2 \cdot \delta_E(U, V) = \delta_D(U_\$, V_\$)$.*

Note that an LCP query on suffixes of $U_\$$ and $V_\$$ trivially reduces in $\mathcal{O}(1)$ time to an LCP query on suffixes of U and V .

► **Definition 25.** *For two strings U and V and an integer range I , we define the alignment graph $G(U, V, I)$ as the weighted undirected graph over the set of vertices \mathbb{Z}^2 with the following edges for each $(a, b) \in \mathbb{Z}^2$:*

- $(a, b) \leftrightarrow (a+1, b)$ with weight 1, $(a, b) \leftrightarrow (a, b+1)$ with weight 1,
- $(a, b) \leftrightarrow (a+1, b+1)$ with weight 0 unless $a \in [0..|U|)$, $b \in [0..|V|)$, $U[a] \neq V[b]$, and $b-a \in I$.

For an alignment graph $G(U, V, I)$, there are $|I|$ diagonals on which diagonal edges may be missing. Intuitively, everything outside these diagonals is considered as a match.

► **Observation 26** ([16, see Lemma 8.5]). *For all fragments $U[a..a']$ and $V[b..b']$ of U and V , respectively, $\delta_D(U[a..a'], V[b..b'])$ is the length of the shortest $(a, b) \rightsquigarrow (a', b')$ path in $G(U, V, \mathbb{Z})$.*

Let $G := G(P, T, [p-k..p+2k])$. For each $t \in [0..m]$, we define a $(3k+2) \times (3k+2)$ distance matrix D_t such that $D_t[i, j]$ is the length of the shortest path between $(0, i+p-k-1)$ and $(t, t+j+p-k-1)$ in the alignment graph G . Let us recall that a matrix M is a *Monge matrix* if for every pair of rows $i < j$ and every pair of columns $\ell < r$, $M[i, \ell] + M[j, r] \leq M[i, r] + M[j, \ell]$. The planarity of G and the fact that all vertices of the considered shortest paths lie in $[0..t] \times \mathbb{Z}$, imply the following.

► **Observation 27.** *For every $t \in [0..m]$, the matrix D_t is a Monge matrix.*

Let us recall that a permutation matrix is a square matrix over $\{0, 1\}$ that contains exactly one 1 in each row and in each column. A permutation matrix P of size $s \times s$ corresponds to a permutation π of $[0..s)$ such that $P[i, j] = 1$ if and only if $\pi(i) = j$. For two permutations π_1, π_2 and their corresponding permutation matrices P_1, P_2 , by $\Delta(\pi_1, \pi_2) = \Delta(P_1, P_2)$ we

denote a shortest sequence of transpositions f_1, \dots, f_q of neighboring elements such that $f_q \circ \dots \circ f_1 \circ \pi_1 = \pi_2$. For an $s \times s$ matrix A , we denote by A^Σ an $(s+1) \times (s+1)$ matrix such that

$$A^\Sigma[i, j] = \sum_{i' \geq i} \sum_{j' < j} A[i', j'], \quad \text{for } i, j \in [0..s].$$

The lemma below follows readily from the tools developed in [16, Sections 8, 9] which, in turn, are based on the ideas of Tiskin [44, 45].

► **Lemma 28.**

- (a) Let $t \in [0..m]$, $i \in [p..p+k]$, $j \in [i-k..i+k] \cap [i-t..n-t]$, and $k' \in [0..k]$. Then, $\delta_D(T[i..j+t], P[0..t]) \leq k'$ if and only if $D_t[i-p+k+1, j-p+k+1] \leq k'$.
- (b) For each $t \in [0..m]$, there is a $(3k+1) \times (3k+1)$ permutation matrix P_t such that $D_t[i, j] = 2P_t^\Sigma[i, j] + i - j$ holds for all $i, j \in [0..3k+1]$. P_0 is an identity permutation matrix. A sequence $\Delta(P_0, P_1), \dots, \Delta(P_{m-1}, P_m)$ contains at most $3k(3k+1)/2$ transpositions of neighboring elements in total and all its non-empty elements can be computed in $\mathcal{O}(k^2 \log \log k)$ time plus $\mathcal{O}(k^2)$ LCP queries on pairs of suffixes of P and T .

In the following lemma, whose proof is omitted here, we extend a known result on answering submatrix maximum queries on Monge matrices [35] (see also [25]) to a dynamic (the matrix changes by sub-row increments) and partially persistent (we need to be able to query all previously created matrices) setting. Sub-column queries are sufficient for our purposes; such queries were considered as a simpler case also in [35, 25]. We further consider minimum queries instead of maximum queries, which is a fairly straightforward change.

► **Lemma 29.** Let $M_0 = M$ be an $s \times s$ Monge matrix such that each entry of M_0 can be retrieved in $\mathcal{O}(1)$ time. We consider a sequence of operations:

- A sub-row increment takes the most recent matrix M_i and creates a matrix M_{i+1} resulting after this operation.
- A sub-column minimum query extracts the minimum in a given sub-column of a specified previously created matrix M_i .

The data structure for M_0 can be initialized in $\mathcal{O}(s \log^2 s)$ time. If each created matrix is a Monge matrix, then each incrementation can be performed in $\mathcal{O}(\log^3 s)$ time and each query can be answered in $\mathcal{O}(\log^2 s)$ time.

Proof of Theorem 4. Let us recall that we make all computations for $\mathcal{O}(n/k)$ values of p . We will store D_t for $t \in [0..m]$ using the data structure of Lemma 29. The initialization of D_0 takes $\mathcal{O}(k \log^2 k)$ time; we have $D_0[a, b] = |a - b|$. Each transposition in the maintained matrix P_t corresponds to a sub-column increment in D_t . Note that, for each $t \in [0..m]$, D_t is a Monge matrix due to Observation 27. Any intermediate matrix D is also Monge as it satisfies $D[i, j] = 2P^\Sigma[i, j] + i - j$ for a (maintained) permutation matrix P . Thus, for each t such that $\Delta(P_t, P_{t+1}) \neq \emptyset$, we can update the maintained Monge matrix as necessary using Lemma 29. By Lemma 28(b), the number of updates will be $\mathcal{O}(k^2)$ and the updates can be computed in $\mathcal{O}(k^2 \log \log k)$ time after $\mathcal{O}(n)$ -time preprocessing for LCP queries. The updates are performed in $\mathcal{O}(k^2 \log^3 k)$ total time.

We are to compute, for all $k' \in [0..k]$ and $i \in [p..p+k]$, the length of the longest prefix of P that matches a prefix of $T[i..n]$ with deletion distance at most k' . By Lemma 28(a) it suffices to find the maximum $t \in [0..m]$ such that $\min\{D_t[i-p+k+1, j-p+k+1] : j \in [i-k..i+k] \cap [i-t..n-t]\} \leq k'$. To this end, we apply binary search with $\mathcal{O}(\log k)$ steps on the set of all t that satisfy $\Delta(P_t, P_{t+1}) \neq \emptyset$, using $\mathcal{O}(\log^2 k)$ -time queries of Lemma 29. Thus, all binary searches take $\mathcal{O}(k^2 \log^3 k)$ time in total. The total time is $\mathcal{O}(n + (n/k) \cdot k^2 \log^3 k) = \mathcal{O}(nk \log^3 k)$. ◀

5 Conditional Hardness of Approximate CPM

We consider the following problem where the number of allowed mismatches is unbounded.

MISMATCH-CPM
Input: A text T of length n and a pattern P of length m .
Output: An array $\text{CPM}[0..n-m]$ with $\text{CPM}[i] = \min\{k \geq 0 : P \approx_k^{\delta_H} T[i..i+m]\}$.

In jumbled indexing, we are to answer pattern matching queries in the jumbled (abelian) sense. More precisely, given a Parikh vector of a pattern that specifies the quantity of each letter, we are to check if there is a substring of the text with this Parikh vector. In the case of a binary text, the problem of constructing a jumbled index is known to be equivalent (up to a log n -factor in the case where a concrete substring needs to be identified; see [20]) to constructing the following data structure.

Given a text X of length n over alphabet $\{0, 1\}$, for each $t \in [1..n]$ compute the values:

$$\min_t := \min \left\{ \sum_{j=i}^{i+t-1} X[j] : i \in [0..n-t] \right\}, \max_t := \max \left\{ \sum_{j=i}^{i+t-1} X[j] : i \in [0..n-t] \right\}.$$

For a few years since its introduction [19] the problem of constructing a binary jumbled index in $\mathcal{O}(n^{2-\epsilon})$ time for $\epsilon > 0$ was open. Chan and Lewenstein [13] settled this question affirmatively by proposing an $\mathcal{O}(n^{1.859})$ -time randomized construction; very recently it was improved to $\mathcal{O}(n^{1.5} \log^{\mathcal{O}(1)} n)$ time [18]. We make the following reduction.

► **Theorem 30.** *If Mismatch-CPM on binary strings can be solved in $S(n)$ time, then the BJI can be constructed in $\mathcal{O}(n + S(3n))$ time.*

Proof. We show how to compute \max_t for all $t \in [1..n]$. For computing \min_t , we can negate all the letters of X . An illustration of our reduction is provided in Figure 4.

$X = 0100101001001$	$n = 13$	$j = 5$	$t = 6$	$i = n - t = 7$
$\min_1 = 0$	$\max_1 = 1$	$\text{CPM}[i] = t - \max_t + j - \max_t = \text{CPM}[7] = 5$		
$\min_2 = 0$	$\max_2 = 1$	$P = 01001010010010000000000000$		
$\min_3 = 1$	$\max_3 = 2$	$T = \overset{0}{1}111111\overset{7}{1}11111000000000000000000000000000$		
$\min_4 = 1$	$\max_4 = 2$	$\text{rot}_4(P) = 101001001000000000000000100$		
$\min_5 = 1$	$\max_5 = 2$			
$\min_6 = 2$	$\max_6 = 3$			
\dots				

■ **Figure 4** $\text{CPM}[7] = 5$ corresponds to a Hamming distance of 5 between $T[7..7+|P|)$ and some rotation of P , namely of $\text{rot}_4(P)$.

It suffices to consider an instance of Mismatch-CPM with $P = X0^n$, $T = 1^n0^{2n}$. A prefix of a rotation of P of length at most n is also a substring of a string 0^nX0^n . For each $i \in [0..n)$, to compute $\text{CPM}[i]$ we need to choose a substring of 0^nX0^n of length $t = n - i$ with the most number of 1s as the prefix of some rotation of P . Indeed, if U is this prefix and P' is this rotation, that is, $P' = UV$, the remaining 0s in U and 1s in V will correspond to

mismatches between P' and $T[i..i+|P|)$. The maximum number of 1s among the substrings of $0^n X 0^n$ is the same as the maximum for X since it is never worse to choose a length- t prefix (suffix) of X than a shorter prefix (suffix) and a part from 0^n .

Thus, we have $\text{CPM}[i] = t - \max_t + j - \max_t$, where j is equal to the number of 1s in X . Hence, $\max_t = (t + j - \text{CPM}[n - t])/2$. These values can be recovered from the output of Mismatch-CPM in linear time. ◀

The following theorem shows how to compute the edit distance of two strings with the use of an algorithm for the decision version of k -Edit CPM, and thus also that a strongly subquadratic such algorithm would refute SETH [34]. The proof is omitted in this version.

► **Theorem 31.** *If k -Edit CPM on quaternary strings can be solved in $\mathcal{O}(n^{2-\varepsilon})$ time for some constant $\varepsilon > 0$, then the edit distance of two binary strings each of length at most n can be computed in $\mathcal{O}(n^{2-\varepsilon} \log n)$ time.*

References

- 1 Karl R. Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16(6):1039–1051, 1987. doi:10.1137/0216067.
- 2 Lorraine A. K. Ayad, Carl Barton, and Solon P. Pissis. A faster and more accurate heuristic for cyclic edit distance computation. *Pattern Recognition Letters*, 88:81–87, 2017. doi:10.1016/j.patrec.2017.01.018.
- 3 Lorraine A. K. Ayad and Solon P. Pissis. MARS: Improving multiple circular sequence alignment using refined sequences. *BMC Genomics*, 18(1):86, 2017. doi:10.1186/s12864-016-3477-5.
- 4 Lorraine A. K. Ayad, Solon P. Pissis, and Ahmad Retha. libFLASM: A software library for fixed-length approximate string matching. *BMC Bioinformatics*, 17:454:1–454:12, 2016. doi:10.1186/s12859-016-1320-2.
- 5 Md. Aashikur Rahman Azim, Costas S. Iliopoulos, M. Sohel Rahman, and M. Samiruzzaman. A filter-based approach for approximate circular pattern matching. In *Bioinformatics Research and Applications - 11th International Symposium, ISBRA 2015*, volume 9096 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2015. doi:10.1007/978-3-319-19048-8_3.
- 6 Carl Barton, Costas S. Iliopoulos, Ritu Kundu, Solon P. Pissis, Ahmad Retha, and Fatima Vayani. Accurate and efficient methods to improve multiple circular sequence alignment. In *Experimental Algorithms, SEA 2015*, volume 9125 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 2015. doi:10.1007/978-3-319-20086-6_19.
- 7 Carl Barton, Costas S. Iliopoulos, and Solon P. Pissis. Fast algorithms for approximate circular string matching. *Algorithms for Molecular Biology*, 9:9, 2014. doi:10.1186/1748-7188-9-9.
- 8 Carl Barton, Costas S. Iliopoulos, and Solon P. Pissis. Average-case optimal approximate circular string matching. In *Language and Automata Theory and Applications - 9th International Conference, LATA 2015*, volume 8977 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 2015. doi:10.1007/978-3-319-15579-1_6.
- 9 Carl Barton, Costas S. Iliopoulos, Solon P. Pissis, and William F. Smyth. Fast and simple computations using prefix tables under Hamming and edit distance. In *Combinatorial Algorithms - 25th International Workshop, IWOCA 2014*, volume 8986 of *Lecture Notes in Computer Science*, pages 49–61. Springer, 2014. doi:10.1007/978-3-319-19315-1_5.
- 10 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.
- 11 Gerth Stølting Brodal, Pooya Davoodi, and S. Srinivasa Rao. Path minima queries in dynamic weighted trees. In *Algorithms and Data Structures - 12th International Symposium, WADS 2011*, volume 6844 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2011. doi:10.1007/978-3-642-22300-6_25.

- 12 Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern Hamming distances. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 643–656. ACM, 2020. doi:10.1145/3357713.3384266.
- 13 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568.
- 14 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, 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.
- 15 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 978–989. IEEE, 2020. doi:10.1109/FOCS46700.2020.00095.
- 16 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster pattern matching under edit distance. In *Accepted to 63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, 2022. arXiv:2204.03087.
- 17 Kuei-Hao Chen, Guan-Shieng Huang, and Richard Chia-Tung Lee. Bit-parallel algorithms for exact circular string matching. *The Computer Journal*, 57(5):731–743, 2014. doi:10.1093/comjnl/bxt023.
- 18 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC 2022: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.
- 19 Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Searching for jumbled patterns in strings. In *Proceedings of the Prague Stringology Conference 2009*, pages 105–117, 2009. URL: <http://www.stringology.org/event/2009/p10.html>.
- 20 Ferdinando Cicalese, Travis Gagie, Emanuele Giaquinta, Eduardo Sany Laber, Zsuzsanna Lipták, Romeo Rizzi, and Alexandru I. Tomescu. Indexes for jumbled pattern matching in strings, trees and graphs. In *String Processing and Information Retrieval - 20th International Symposium, SPIRE 2013*, volume 8214 of *Lecture Notes in Computer Science*, pages 56–63. Springer, 2013. doi:10.1007/978-3-319-02432-5_10.
- 21 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k -mismatch problem revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 2039–2052. SIAM, 2016. doi:10.1137/1.9781611974331.ch142.
- 22 Michael J. Fischer and Michael S. Paterson. String-matching and other products. Technical report, Massachusetts Institute of Technology, USA, 1974.
- 23 Kimmo Fredriksson and Szymon Grabowski. Average-optimal string matching. *Journal of Discrete Algorithms*, 7(4):579–594, 2009. doi:10.1016/j.jda.2008.09.001.
- 24 Kimmo Fredriksson and Gonzalo Navarro. Average-optimal single and multiple approximate string matching. *ACM Journal of Experimental Algorithmics*, 9, 2004. doi:10.1145/1005813.1041513.
- 25 Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Submatrix maximum queries in Monge and partial Monge matrices are equivalent to predecessor search. *ACM Transactions on Algorithms*, 16(2):16:1–16:24, 2020. doi:10.1145/3381416.
- 26 Paweł Gawrychowski and Przemysław Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPICs*, pages 62:1–62:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.62.

- 27 Roberto Grossi, Costas S. Iliopoulos, Robert Mercas, Nadia Pisanti, Solon P. Pissis, Ahmad Retha, and Fatima Vayani. Circular sequence comparison: algorithms and applications. *Algorithms for Molecular Biology*, 11:12, 2016. doi:10.1186/s13015-016-0076-6.
- 28 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.
- 29 Tommi Hirvola and Jorma Tarhio. Bit-parallel approximate matching of circular strings with k mismatches. *ACM Journal of Experimental Algorithmics*, 22, 2017. doi:10.1145/3129536.
- 30 ThienLuan Ho, Seungrohk Oh, and Hyunjin Kim. New algorithms for fixed-length approximate string matching and approximate circular string matching under the Hamming distance. *The Journal of Supercomputing*, 74(5):1815–1834, 2018. doi:10.1007/s11227-017-2192-6.
- 31 Wing-Kai Hon, Tsung-Han Ku, Rahul Shah, and Sharma V. Thankachan. Space-efficient construction algorithm for the circular suffix tree. In *Combinatorial Pattern Matching, 24th Annual Symposium, CPM 2013*, volume 7922 of *Lecture Notes in Computer Science*, pages 142–152. Springer, 2013. doi:10.1007/978-3-642-38905-4_15.
- 32 Wing-Kai Hon, Chen-Hua Lu, Rahul Shah, and Sharma V. Thankachan. Succinct indexes for circular patterns. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011*, volume 7074 of *Lecture Notes in Computer Science*, pages 673–682. Springer, 2011. doi:10.1007/978-3-642-25591-5_69.
- 33 Costas S. Iliopoulos, Solon P. Pissis, and M. Sohel Rahman. Searching and indexing circular patterns. In *Algorithms for Next-Generation Sequencing Data, Techniques, Approaches, and Applications*, pages 77–90. Springer, 2017. doi:10.1007/978-3-319-59826-0_3.
- 34 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.
- 35 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in Monge matrices and Monge partial matrices, and their applications. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 338–355. SIAM, 2012. doi:10.1137/1.9781611973099.31.
- 36 S. Rao Kosaraju. Efficient string matching. Manuscript, 1987.
- 37 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998. doi:10.1137/S0097539794264810.
- 38 Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989. doi:10.1016/0196-6774(89)90010-2.
- 39 M. Lothaire. *Applied Combinatorics on Words*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2005. doi:10.1017/CB09781107341005.
- 40 Vicente Palazón-González and Andrés Marzal. On the dynamic time warping of cyclic sequences for shape retrieval. *Image and Vision Computing*, 30(12):978–990, 2012. doi:10.1016/j.imavis.2012.08.012.
- 41 Vicente Palazón-González and Andrés Marzal. Speeding up the cyclic edit distance using LAESA with early abandon. *Pattern Recognition Letters*, 62:1–7, 2015. doi:10.1016/j.patrec.2015.04.013.
- 42 Vicente Palazón-González, Andrés Marzal, and Juan Miguel Vilar. On hidden Markov models and cyclic strings for shape recognition. *Pattern Recognition*, 47(7):2490–2504, 2014. doi:10.1016/j.patcog.2014.01.018.
- 43 Robert Susik, Szymon Grabowski, and Sebastian Deorowicz. Fast and simple circular pattern matching. In *Man-Machine Interactions 3, Proceedings of the 3rd International Conference on Man-Machine Interactions, ICMMI 2013*, volume 242 of *Advances in Intelligent Systems and Computing*, pages 537–544. Springer, 2013. doi:10.1007/978-3-319-02309-0_59.
- 44 Alexander Tiskin. Semi-local string comparison: algorithmic techniques and applications, 2007. doi:10.48550/ARXIV.0707.3619.
- 45 Alexander Tiskin. Semi-local string comparison: Algorithmic techniques and applications. *Mathematics in Computer Science*, 1(4):571–603, 2008. doi:10.1007/s11786-007-0033-3.
- 46 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

Multi-Dimensional Stable Roommates in 2-Dimensional Euclidean Space

Jiehua Chen ✉

TU Wien, Austria

Sanjukta Roy ✉

Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

TU Wien, Austria

Abstract

We investigate the EUCLIDEAN d -DIMENSIONAL STABLE ROOMMATES problem, which asks whether a given set V of $d \cdot n$ points from the 2-dimensional Euclidean space can be partitioned into n disjoint (unordered) subsets $\Pi = \{V_1, \dots, V_n\}$ with $|V_i| = d$ for each $V_i \in \Pi$ such that Π is *stable*. Here, *stability* means that no point subset $W \subseteq V$ is blocking Π , and W is said to be *blocking* Π if $|W| = d$ such that $\sum_{w' \in W} \delta(w, w') < \sum_{v \in \Pi(w)} \delta(w, v)$ holds for each point $w \in W$, where $\Pi(w)$ denotes the subset $V_i \in \Pi$ which contains w and $\delta(a, b)$ denotes the Euclidean distance between points a and b . Complementing the existing known polynomial-time result for $d = 2$, we show that such polynomial-time algorithms cannot exist for any fixed number $d \geq 3$ unless $P=NP$. Our result for $d = 3$ answers a decade-long open question in the theory of Stable Matching and Hedonic Games [18, 1, 10, 26, 21].

2012 ACM Subject Classification Theory of computation \rightarrow Problems, reductions and completeness; Theory of computation \rightarrow Solution concepts in game theory; Theory of computation \rightarrow Computational geometry

Keywords and phrases stable matchings, multidimensional stable roommates, Euclidean preferences, coalition formation games, stable cores, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.36

Related Version *Full Version:* <https://arxiv.org/abs/2108.03868>

Funding *Jiehua Chen:* Vienna Science and Technology Fund (WWTF) grant VRG18-012.

Sanjukta Roy: This work was done when SR was affiliated with TU Vienna, and was supported by Vienna Science and Technology Fund (WWTF) grant VRG18-012.

1 Introduction

We study the computational complexity of a geometric and multi-dimensional variant of the classical stable matching problem, called EUCLIDEAN d -DIMENSIONAL STABLE ROOMMATES (EUCLID- d -SR). This problem is to decide whether a given set V of $d \cdot n$ agents, each represented by a point in the two-dimensional Euclidean space \mathcal{R}^2 , has a d -dimensional *stable matching* (in short, d -stable matching). Here, each agent $x \in V$ has a preference list over all (unordered) size- d agent sets containing x which is derived from the Euclidean distances between the points. More precisely, agent x *prefers* subset S to subset T if the sum of Euclidean distances from x to S is smaller than the sum of the distances to T . We call preferences over subsets of agents which are based on the sum of Euclidean distances *Euclidean preferences*. A d -dimensional matching is a partition of V into n disjoint agent subsets $\Pi = \{V_1, \dots, V_n\}$ with $|V_i| = d$ for all $i \in \{1, \dots, n\}$. In this way, each agent $v \in V$ is assigned to a subset in Π . An agent subset V' is *blocking* the d -dimensional matching Π if $|V'| = d$ and each agent in V' prefers V' to its “assigned” agent subset in Π . A d -stable matchings is a d -matchings that is not blocked by a subset of agents of size d .



© Jiehua Chen and Sanjukta Roy;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 36; pp. 36:1–36:16
Leibniz International Proceedings in Informatics



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

When allowing agents to have arbitrary preferences, we arrive at the d -DIMENSIONAL STABLE ROOMMATES (d -SR) problem with 2-SR being equivalent to the classical STABLE ROOMMATES problem [14, 17]. It is well-known that *not* every instance of STABLE ROOMMATES admits a 2-stable matching, but deciding whether there exists one is polynomial-time solvable [17]. Fortunately, if we restrict the preferences to be Euclidean, then a 2-stable matching always exists and it can be found in polynomial time: Iteratively pick two remaining agents who are closest to each other and match them [1]. One may be tempted to apply this greedy approach to the case when $d = 3$. However, this would only work if it can find and match a triple of agents in each step such that this triple is the most preferred one of all three. Since such a “most-preferred” triple may not always exist, the prospects become less clear. Indeed, Arkin et al. [1] showed that not every instance of EUCLID-3-SR admits a 3-stable matching. To the best of our knowledge, nothing about the existence of EUCLID- d -SR is known for any fixed $d \geq 4$. In particular, the no instance by Arkin et al. will not work for any fixed $d \geq 4$. Arkin et al. left open the computational complexity of finding a 3-stable matching. The same question has been repeatedly asked since then [18, 21, 10, 26, 6]. Nevertheless, d -SR (i.e., for general preferences) has been known to be NP-complete for $d = 3$. Hence, it is of particular importance to search for natural restricted subcases, e.g., under Euclidean preferences, which may allow for efficient algorithms.

Our contribution. In this work, we aim at settling the computational complexity of EUCLID- d -SR for all fixed $d \geq 3$. Arkin et al. [1] showed that there is always a 3-dimensional matching which is approximately stable, which sparks hope for a polynomial-time algorithm for $d = 3$. We destroy such hope by showing that EUCLID-3-SR is NP-hard. We achieve this by reducing from an NP-complete planar variant of the EXACT COVER BY 3 SETS problem, where we make use of a novel chain gadget (see the orange and blue parts in Figure 3) and a star gadget (see Figure 1) which is adapted from the no-instance of Arkin et al. See the idea part in Section 3 for more details.

The same construction does not work for $d \geq 4$ since a no-instance for EUCLID-3-SR does not remain a no-instance for EUCLID-4-SR. However, we manage to derive two extended star structures, one for odd d and the other for even d (see the right and left figures of Figure 4, respectively), adapt the remaining component gadgets to show hardness for all fixed $d \geq 4$. Together, we show the following.

► **Theorem 1.** EUCLID- d -SR is NP-complete for every fixed $d \geq 3$.

Related work. Knuth [19] proposed to generalize the well-known STABLE MARRIAGE problem (a bipartite restriction of the STABLE ROOMMATES problem) to the 3-dimensional case. There are many such generalized variants in the literature, including the NP-complete 3-SR problem [18]. Huang [16] strengthen the result by showing that 3-SR remains NP-hard even for *additive* preferences. Herein, each agent $x \in V$ has cardinal preferences $\mu_x: V \setminus \{x\} \rightarrow \mathbb{R}$ over all other agents such that x prefers $\{x, s_1, s_2\}$ to $\{x, t_1, t_2\}$ if and only if $\mu_x(s_1) + \mu_x(s_2) > \mu_x(t_1) + \mu_x(t_2)$. Deineko and Woeginger [10] strengthen the result of Huang by showing that 3-SR remains NP-hard even for metric preferences: $\mu_x(y) = \mu_y(x) \geq 0$ and $\mu_x(y) + \mu_y(z) \leq \mu_x(z)$ such that x prefers $\{x, s_1, s_2\}$ to $\{x, t_1, t_2\}$ if and only if $\mu_x(s_1) + \mu_x(s_2) < \mu_x(t_1) + \mu_x(t_2)$. It is straightforward to see that Euclidean preferences are metric preferences and metric preferences are additive. We thus strengthen the results of Deineko and Woeginger, and Huang, by showing that the hardness remains even for Euclidean preferences. Recently, McKay and Manlove [22] strengthen the result of Huang [16] by showing that the NP-hardness remains even if the cardinal preferences are binary, i.e.,

$\mu_x(y) \in \{0, 1\}$ for all other agents y . This result is not comparable to ours since binary preferences and Euclidean preferences are not comparable. They also show that 3-SR becomes polynomial-time solvable when the preferences are binary and symmetric.

Multi-dimensional stable matchings are equivalent to the so-called *fixed-size stable cores* in hedonic games [12], where each coalition (i.e., a non-empty subset of agents) in the core must have the same size, and stability only needs to be guaranteed for any other coalition of the same size.¹ Hence, our NP-hardness result also transfers to the case of finding a fixed-size stable core in the scenario where the agents in the hedonic game have Euclidean preferences. Hedonic games have been studied under graphical preference models [11, 24], where there is an underlying social network (a directed graph) such that agents correspond to the vertices in the graph. The general idea is to assume that agents prefer to be with their own out-neighbors more than non-out-neighbors. The Euclidean preference model is related to the graphical preference model where the underlying graph is planar. However, the Euclidean model is more fine-grained and assumes that the intensity of the preferences also depends on the distance of the agents. Notably, under the graphical model, a stable core always exists and it can be found in linear time [11], but verifying whether a given partition is stable is NP-hard [7]. Hedonic games with fixed-size coalitions have been studied for other solution concepts such as strategy-proofness [27], Pareto optimality [9], and exchange stability [3].

Other generalized variants include the study of 3-stable matching with cyclic preferences [13, 4, 20], with preferences over individuals [18], and the study of the higher-dimensional case [6] and of other restricted preference domains [5]. We refer to the textbook by Manlove [21] for more references.

Paper outline. In Section 2, besides introducing necessary concepts and notations used throughout the paper, we describe a crucial star-structured instance of EUCLID-3-SR (see Example 2), which serves as a tool of our NP-hardness reduction. The proof of Theorem 1 is divided into two sections: In Section 3, we consider the case of $d = 3$ and show-case in detail how to combine the star-structured instance with two new gadgets, one for the local replacement and one for the enforcement, to obtain NP-hardness. In Section 4, we show how to carefully adapt the star-structured instance (which only works for $d = 3$) and modify the reduction to show hardness for any fixed $d \geq 4$. We conclude in Section 5. Due to space constraints, some figures, examples, and (part of) the proofs for results marked by \star are deferred to the full version [8].

2 Preliminaries

Given a non-negative integer t , we use “[t]” (without any prefix) to denote the set $\{1, \dots, t\}$. Throughout the paper, if not stated explicitly, we assume that ε and ε_d are small fractional values with $0 < \varepsilon < 0.001$ and $0 < \varepsilon_d < \frac{1}{1000d}$, where $d \geq 3$. By “close to zero” we mean a value which is smaller than ε and ε_d .

For each fixed integer $d \geq 2$, an instance of EUCLIDEAN d -DIMENSIONAL STABLE ROOMMATES (EUCLID- d -SR) consists of a set $V = \{1, \dots, d \cdot n\}$ of $d \cdot n$ agents and an embedding $E: V \rightarrow \mathbb{R}^d$ of the agents into d -dimensional Euclidean space. We call a non-empty subset $V' \subseteq V$ of agents a *coalition*. The preference list \succeq_x of each agent $x \in V$ over

¹ A stable core is a *partition* Π of the agents into disjoint coalitions such that no subset of agents would block the partition Π by forming its own new coalition.

all possible size- d coalitions containing x is derived from the sum of the Euclidean distances from x to the coalition. More precisely, for each two size- d coalitions $S = \{x, a_1, \dots, a_{d-1}\}$ and $T = \{x, b_1, \dots, b_{d-1}\}$ containing x we say that x *weakly prefers* S to T , denoted as $S \succeq_x T$, if the following holds:

$$\sum_{j \in [d-1]} \delta(E(x), E(a_j)) \leq \sum_{j \in [d-1]} \delta(E(x), E(b_j)),$$

where $\delta(p, q) := \sqrt{(p[1] - q[1])^2 + (p[2] - q[2])^2}$. We use $S \succ_x T$ (i.e., x preferring S to T) and $S \sim_x T$ (i.e., x indifferent between S and T) to refer to the asymmetric and symmetric part of \succeq_x , respectively. To ease notation, for an agent x and a preference list \mathcal{L} over a subset \mathcal{F} of size- d coalitions, we use $\mathcal{L} \succ_x \dots$ to indicate that agent x prefers every size- d coalition in \mathcal{F} over every size- d coalition not in \mathcal{F} and her preferences over \mathcal{F} are according to \mathcal{L} . Further, we use the agent and her embedded points interchangeably, and the distance between two agents means the distance between their embedded points. For each agent x and each coalition $S \subseteq V$, we use $\delta(x, S)$ to refer to the sum of Euclidean distances from x to each member in S : $\delta(x, S) = \sum_{y \in S} \delta(x, y)$.

See the introduction for the definition of *d-matchings*, *blocking coalitions*, and *d-stable matchings*. Given a d -matching Π and an agent $x \in V$, let $\Pi(x)$ denote the coalition that contains x . The problem studied in this paper is defined as follows:

EUCLID- d -SR

Input: An agent set $V = \{1, \dots, d \cdot n\}$ and an embedding $E: V \rightarrow \mathbb{R}^2$.

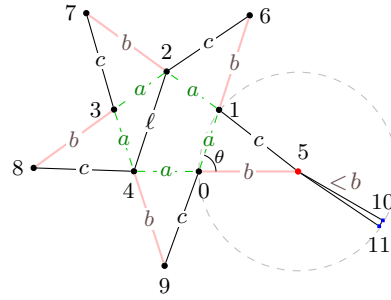
Question: Is there a d -stable matching?

Note that since stability for each fixed d can be checked in polynomial time, EUCLID- d -SR is contained in NP for every fixed d .

Not every EUCLID-3-SR instance admits a 3-stable matching. Arkin et al. [1] provided a star-structured instance which does not. In Example 2, we describe an *adapted* variant of their instance, which is a decisive component of our hardness reduction.

► **Example 2.** Consider an instance which contains *at least* 12 agents called $W = \{0, \dots, 11\}$ where the 12 agents are embedded as given in Figure 1. In the embedding of $W \setminus \{10, 11\}$, the five inner-most points, namely 0 to 4, form a regular pentagon with edge length a . For each $i \in \{0, \dots, 4\}$, the three points $i, i + 1 \bmod 5$, and $i + 5$ form a triangle with side lengths a, b, c such that $a < b < c < \ell$, where ℓ denotes the diagonal of the regular pentagon. Moreover, the angle θ at points $i + 1 \bmod 5, i, i + 5$ is at most 90 degrees. This ensures that the distance between points $(i + 1 \bmod 5) + 5$ and i is strictly larger than ℓ (we will use this fact later). Except for point 5 (marked in red), the closest neighbor of each point $i + 5$ is i , followed by $i + 1 \bmod 5$. Point 5's two closest neighbors are points 10 and 11 with $a < \delta(5, 10) < b$ and $a < \delta(5, 11) < b$, followed by points 0 and 1. The distance between 10 and 11 is close to zero, with the intention to ensure that every 3-stable matching must match them together. The distance from 10 (resp. 11) to any agent in $W \setminus \{5, 10, 11\}$ is larger than the diagonal length ℓ while the distance from 10 (resp. 11) to any agent not in W is larger than $\delta(5, 10) - \varepsilon$. Finally, the distance between any agent from $W \setminus \{10, 11\}$ to any agent *not* from $W \setminus \{10, 11\}$ is strictly larger than ℓ .

To specify the embedding of the agents from W , we use the polar coordinate system. We first fix the embeddings of 5, 10, 11 to ensure the distances between them are as stated above. Then, we fix points 0 and 1 and the centroid of the regular pentagon to ensure the distances satisfy $a < b < c < \ell$, and the angle θ at points 1, 0, 5 is at most 90 degrees, and the angle at points 0, 5, $j, j \in \{10, 11\}$, is more than 90 degrees. Once these points are fixed we can determine the other points by a simple calculation.



■ **Figure 1** A star-structured instance adapted from Arkin et al. [1]; see Example 2. We use different colors to highlight the distances between the points. For instance, the smallest distance between any two points is a (highlighted in green). We also draw a dashed circle of radius b , centered at point 5 to indicate that points both 10 and 11 are with distance smaller than b to 5.

The instance of Arkin et al. [1] embeds the two extra points 10 and 11 differently than ours (see Example 2). Hence, their instance is a no-instance, while ours may be a yes-instance, provided some specific triple is matched together, formulated as follows:

► **Lemma 3.** *Every 3-stable matching of an instance satisfying the embedding described in Example 2 must contain triple $\{5, 10, 11\}$.*

Proof. Towards a contradiction, suppose that Π is a 3-stable matching with $\{5, 10, 11\} \notin \Pi$. We infer that $\{10, 11\} \subseteq \Pi(10)$ since otherwise $\{5, 10, 11\}$ is blocking Π due the following: $\delta(5, \Pi(5)) \geq \min(\delta(5, 10), \delta(5, 11)) + b > \delta(5, \{5, 10, 11\})$, and for each $x \in \{10, 11\}$ it holds that $\delta(x, \Pi(x)) \geq 2(\delta(x, 5) - \varepsilon) > \delta(x, 5) + \delta(10, 11)$ for any $\varepsilon > 0$. This implies that $\{10, 11\} \cap \Pi(5) = \emptyset$. Next, we observe that there must be a triple in Π that contains the two agents of at least one pentagon edge as otherwise $\{2, 3, 7\}$ is blocking: $\delta(2, \Pi(2)) \geq b + c > a + b$, $\delta(3, \Pi(3)) \geq b + c > a + c$, and $\delta(7, \Pi(7)) \geq b + \ell > b + c$. Thus, at least one triple in Π contains the agents of some pentagon edge, say $\{2, 3\}$; the other cases are analogous. Let $\{2, 3, x\} \in \Pi$. We distinguish between three subcases:

Case 1: $x \notin \{1, 4, 7, 8\}$. Then, one can verify that $\{2, 3, 7\}$ is blocking; recall that every agent not in $W \setminus \{2, 3\}$ is at distance larger than ℓ to agent 7.

Case 2: $x \in \{1, 7\}$. Then, $\Pi(4) = \{0, 4, 9\}$ or $\Pi(4) = \{0, 4, 8\}$ since otherwise $\{3, 4, 8\}$ blocks Π due to: $\delta(3, \Pi(3)) \geq a + \min(\delta(3, 1), \delta(3, 7)) = a + c > a + b$, $\delta(4, \Pi(4)) > a + c$, $\delta(8, \Pi(8)) > b + c$ (recall that the distance from every agent not in $W \setminus \{3, 4\}$ to agent 8 is larger than ℓ). However, both cases imply that $\{0, 1, 5\}$ is blocking since $\delta(0, \Pi(0)) \geq a + c > a + b = \delta(0, \{0, 1, 5\})$, $\delta(1, \Pi(1)) \geq a + \ell > a + c = \delta(1, \{0, 1, 5\})$, and $\delta(5, \Pi(5)) \geq c + \ell > b + c = \delta(5, \{0, 1, 5\})$; recall that $\Pi(5) \cap \{10, 11\} = \emptyset$.

Case 3: $x \in \{4, 8\}$. Then, $\delta(2, \Pi(2)) \geq a + \ell > a + c$. This implies that $\{0, 1, 6\} \in \Pi$ since otherwise $\{1, 2, 6\}$ is blocking Π . However, this implies that $\{0, 4, 9\}$ is blocking Π .

Since we have just shown that no agent x exists which is in the same triple as 2 and 3, no 3-stable matching exists that does not contain $\{5, 10, 11\}$. ◀

3 NP-hardness for EUCLID-3-SR

In this section, we prove Theorem 1 for the case of $d = 3$ by providing a polynomial reduction from the NP-complete PLANAR AND CUBIC EXACT COVER BY 3 SETS problem [23], which is an NP-complete restricted variant of the EXACT COVER BY 3 SETS problem [15].

PLANAR AND CUBIC EXACT COVER BY 3 SETS (PC-X3C)

Input: A $3n$ -element set $X = \{1, \dots, 3n\}$ and a collection $\mathcal{S} = (S_1, \dots, S_m)$ of 3-element subsets of X of cardinality $3n$ such that each element occurs in exactly three sets and the associated graph is planar.

Question: Does \mathcal{S} contain an *exact cover* for X , i.e., a subcollection $\mathcal{K} \subseteq \mathcal{S}$ such that each element of X occurs in exactly one member of \mathcal{K} ?

Herein, given a PC-X3C instance $I = (X, \mathcal{S})$, the associated graph of I , denoted as $G(I)$, is a bipartite graph $G(I) = (U \uplus W, E)$ on two partite vertex sets $U = \{u_i \mid i \in X\}$ and $W = \{w_j \mid S_j \in \mathcal{S}\}$ such that there exists an edge $e = \{u_i, w_j\} \in E$ if and only if $i \in S_j$. We call the vertices in U and W the *element-vertices* and the *set-vertices*, respectively.

In our reduction, we crucially utilize the fact that the associated graph G of the input instance is planar and cubic, and hence by Valiant [25] admits a specific planar embedding in \mathbb{Z}^2 , called *orthogonal drawing*, which maps each vertex to an integer grid point and each edge to a chain of non-overlapping horizontal and vertical segments along the grid (except at the endpoints). To simplify the description of the reduction, we use the following more restricted orthogonal drawing:

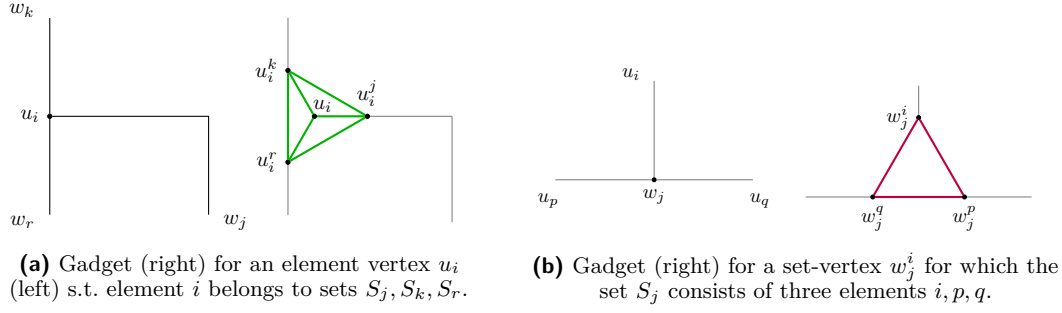
► **Proposition 4** ([2]). *In polynomial time, a planar graph with maximum vertex degree three can be embedded in the grid \mathbb{Z}^2 such that its vertices are at the integer grid points and its edges are drawn using at most one horizontal and one vertical segment in the grid.*

We call the intersection point of the horizontal and vertical segments the *bending point*.

3.1 The construction

The idea. Given an instance $I = (X, \mathcal{S})$ of PC-X3C, we first use Proposition 4 to embed the associated graph $G(I) = (U \uplus W, E)$ into a 2-dimensional grid with edges drawn using line segments of length at least $L \geq 200$, and with parallel lines at least $4L$ grid squares apart. The idea is to replace each element-vertex $u_i \in U$ with four agents which form a “star” with three close-by “leaves” (see Figure 2a). These leaves one-to-one correspond to the sets S_j with $i \in S_j$. In this way, exactly one set S_j is unmatched with the center and will be chosen to the exact cover solution. Furthermore, we replace each set-vertex $w_j \in W$ with three agents w_j^i , $i \in S_j$, which form an equilateral triangle (see Figure 2b). We replace each edge in $G(I)$ with a chain of copies of three agents, which, together with a private enforcement gadget (the star structure with a tail in Figure 3), ensure that either all three agents w_j^i are matched in the same triple (indicating that the corresponding set is in the solution) or none of them is matched in the same triple (indicating that the corresponding set is not in the solution). The agents in the star structure can be embedded “far” from other agents due to the tail.

Gadgets for the elements and the sets. For each element-vertex $u_i \in U$, assume that the three connecting edges in $G(I)$ are going horizontally to the right (rightward), vertically up (upward), and vertically down (downward); we can mirror the coordinate system if this is not the case. Let w_j, w_k, w_r denote the set-vertices on the endpoints of the rightward, upward, and downward edge, respectively. We create four element-agents, called u_i , u_i^j , u_i^k , and u_i^r . We embed them into \mathbb{R}^2 in such a way that u_i^j, u_i^k, u_i^r are on the segment of the rightward, upward, and downward edge, respectively, and are of equal distance δ to each other. Agent u_i is in the center of the other three agents. See Figure 2a for an illustration.



■ **Figure 2** Element- and set-gadgets described in Subsection 3.1.

Similarly, for each set-vertex $w_j \in W$, assume that the three connecting edges in $G(I)$ are going rightward, leftward, and upward, connecting the element-vertices u_i, u_p, u_q , respectively. We create three set-agents, called w_j^i, w_j^p, w_j^q . We embed them into \mathbb{R}^2 in such a way that w_j^i, w_j^p, w_j^q are on the segment of the rightward, leftward, and upward edge, respectively, and are of equidistance $\frac{1}{3}$ to each other. See Figure 2b for an illustration.

The edge- and the enforcement gadget. For each edge $e = \{u_i, w_j\}$ in $G(I)$, we create \hat{n} (a constant value to be determined later) copies of the triple $A_i^j[z] = \{\alpha_i^j[z], \beta_i^j[z], \gamma_i^j[z]\}$, $1 \leq z \leq \hat{n}$, of agents and embed them around the line segments of edge e in the grid (refer to Figure 3). To connect to the set-gadget, we merge agent $\gamma_i^j[\hat{n}]$ and set-agent w_j^i together. For technical reasons, we also use $\gamma_i^j[0]$ to refer to u_i^j . To define the distances, let $\varepsilon_1, \varepsilon_2, \dots, \varepsilon_{2\hat{n}}$ be a sequence of increasing positive values with $2(2\hat{n} - 1)/(2\hat{n} + 1) \leq \varepsilon_{2\hat{n}-1} \leq 2(2\hat{n} - 1)/(2\hat{n}) < \varepsilon_{2\hat{n}} = 2 - \varepsilon$. Now, we embed the newly added agents so that the distances between “consecutive agents” on the line increase with $z \in [\hat{n}]$:

- The distance between agents $\alpha_i^j[z]$ and $\beta_i^j[z]$ (marked in blue) is close to zero.
- The distance between agents $\alpha_i^j[z]$ (resp. $\beta_i^j[z]$) and $\gamma_i^j[z]$ is $8 + \varepsilon_{2z}$.
- The distance between $\alpha_i^j[z]$ (resp. $\beta_i^j[z]$) and $\gamma_i^j[z - 1]$ is $8 + \varepsilon_{2z-1}$.

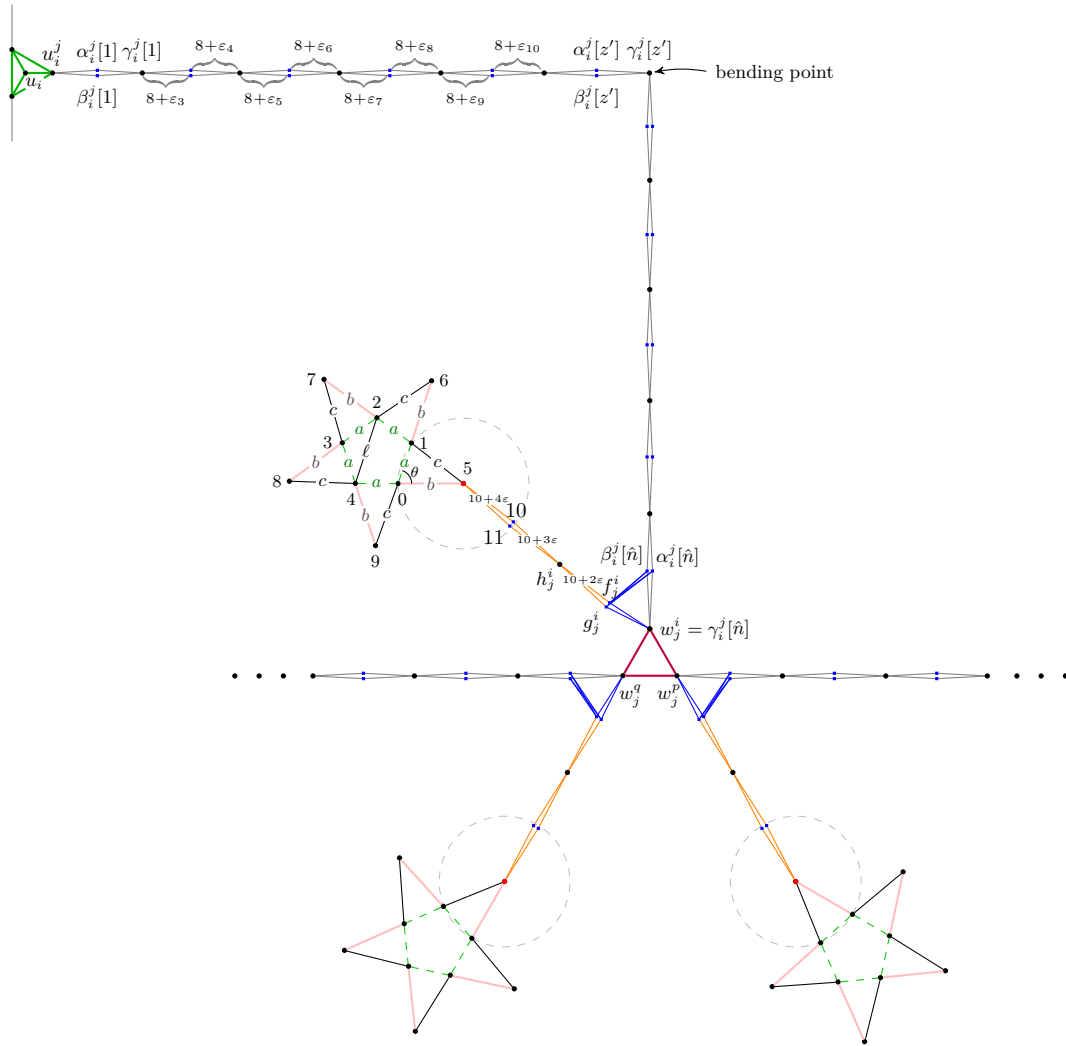
In this manner, we will ensure that either all $A_i^j[z]$, $z \in [\hat{n} - 1]$, or all $\{\gamma_i^j[z - 1], \alpha_i^j[z], \beta_i^j[z]\}$, $i \in [\hat{n}]$ belong to a 3-stable matching (to be proved later).

To determine the value \hat{n} , let the lengths of the segments for edge $\{u_i, w_j\}$ in the orthogonal drawing of graph $G(I)$ be L_1 and L_2 , respectively; L_2 is zero if there is only one straight segment. We set \hat{n} to the largest value satisfying $\sum_{z=1}^{2\hat{n}} (8 + 0.01 \cdot z) \leq L_1 + L_2$, which is clearly a constant. For brevity’s sake, when using \hat{n} , we mean the constant associated to an edge $\{u_i, w_j\}$ in the drawing which will be clear from the context. It is also fairly straightforward to check that one can choose the sequence ε_i so that the bending point of the chain is some agent $\gamma_i^j[z']$, $z' \in [n - 1]$ as shown in Figure 3.

By the construction of the gadgets above, each set-agent w_j^i strictly prefers triple $A_i^j[\hat{n}]$ to triple $\{w_j^i, w_j^p, w_j^q\}$ since $\delta(w_j^i, x) < \frac{1}{3} = \delta(w_j^i, y)$ for all $x \in \{\alpha_i^j[\hat{n}], \beta_i^j[\hat{n}]\}$ and $y \in \{w_j^p, w_j^q\}$; recall that $w_j^i = \gamma_i^j[\hat{n}]$ and $\delta(x, \gamma_i^j[\hat{n}]) = 10 - \varepsilon$. To ensure that exactly one of the two triples is chosen, we make use of the star-gadget from Example 2. More precisely, we introduce an agent triple $H_j^i = \{f_j^i, g_j^i, h_j^i\}$ and embed them in such a way that the distances between two “consecutive” agents on the line towards the star-gadget increase:

- The distance between f_j^i and g_j^i is close to zero.
- The distance between agent h_j^i and each of $\{f_j^i, g_j^i\}$ is $10 + 2\varepsilon$.
- The distance between f_j^i (resp. g_j^i) and each of $A_i^j[\hat{n}]$ is in range $[10 + \varepsilon, 10 + 2\varepsilon)$.

This means that the most preferred triple of agent h_j^i is H_j^i , while both f_j^i and g_j^i prefer triple S to H_j^i where $S = \{f_j^i, g_j^i, x\}$ and $x \in A_i^j[\hat{n}]$.



■ **Figure 3** Gadget for edge $\{u_i, w_j\}$ in $G(I)$ with $S_j = \{i, p, q\}$. Here, the fractional values ε_z satisfy $0 < \varepsilon_1 < \dots < \varepsilon_{\hat{n}} = 2 - \varepsilon$. The star-gadget, adapted from Arkin et al. [1], is described in Example 2. To highlight the distances between the points in the star-gadget, we use different colors. For instance, the smallest distance between any two points in the star is a (highlighted in green). We also draw a dashed circle of radius b , centered at point 5 to indicate that points both 10 and 11 are with distance smaller than b to 5.

Finally, we create 12 agents, namely, $W = \{0, \dots, 11\}$, according to Example 2 such that agents 10 and 11's most preferred triple is $\{10, 11, h_j^i\}$, followed by $\{5, 10, 11\}$. More precisely:

- The distance between agent 10 (resp. agent 11) and h_j^i is $10 + 3\varepsilon$.
- The distance between agent 10 (resp. agent 11) and 5 is $10 + 4\varepsilon$.
- The five agents from $\{0, \dots, 4\}$ form a regular pentagon with edge length a . Each two agents on the pentagon form with a private agent a triangle with edge lengths a (marked in green), b (marked in red), and c . We set $b = 10.1$ and $c = 10.2$. The length of the diagonal of the pentagon is ℓ .

Altogether, the lengths satisfy the relation $a < b < c < \ell$ and the specific angle θ is at most 90 degrees. Due to the chain, including f_j^i , g_j^i , and h_j^i , the distance from every agent not from $W \cup \{h_j^i\}$ to every agent from W is larger than ℓ . We call the gadget, consisting of the

star-agents and the triple H_j^i , the *star-gadget* for set-agent w_j^i and element-agent u_i^j . Figure 3 provides an illustration of how the element-gadget, the set-gadget, and the star-gadget are embedded. Note that since the angle between any two line segments is 90 degrees and the line segment has length at least 200, we can make sure that such embedding is feasible.

This completes the description of the construction, which clearly can be done in polynomial time. In total, we constructed $O(4 \cdot 3n + 3 \cdot 3n + 3 \cdot 2\hat{n} \cdot 3n + 15 \cdot 3n) = O(n)$ agents. Note that we only need to have a good approximation of the embedding of the agents in the star-gadget and the equilateral triangle.

3.2 The correctness proof for $d = 3$

Before we proceed with the correctness proof, we summarize the preferences derived from the embedding via the the following observation.

► **Observation 5** (\star). *For each element $i \in X$ and each set $S_j \in \mathcal{S}$ with $S_j = \{i, p, q\}$, let $0, \dots, 11$ denote the 12 agents in the associated star-gadget. Then, the following holds.*

- (i) *The preference list of each agent $x \in \{10, 11\}$ satisfies $\{h_j^i, 10, 11\} \succ_x \dots$.*
- (ii) *For each triple $B \neq \{h_j^i, f_j^i, g_j^i\}$ with $B \succeq_{h_j^i} \{h_j^i, 10, 11\}$ it holds that $B \cap \{10, 11\} \neq \emptyset$.*
- (iii) *For each agent $x \in \{f_j^i, g_j^i\}$ and each triple $B \neq \{f_j^i, g_j^i, h_j^i\}$:*
 - *If $B = \{f_j^i, g_j^i, y\}$ (where $y \in \{\alpha_i^j[\hat{n}], \beta_j^i[\hat{n}], \gamma_j^i[\hat{n}]\}$), then $B \succ_x \{f_j^i, g_j^i, h_j^i\}$.*
 - *If $B \succeq_x \{f_j^i, g_j^i, h_j^i\}$, then $B = \{f_j^i, g_j^i, y\}$ for some $y \in \{\alpha_i^j[\hat{n}], \beta_j^i[\hat{n}], \gamma_j^i[\hat{n}]\}$.*
- (iv) *For each $z \in [\hat{n}]$ the preference list of agent $\gamma_i^j[z]$ satisfies $\{\alpha_i^j[z], \beta_j^i[z], \gamma_i^j[z]\} \succ_{\gamma_i^j[z]} \dots$.*
- (v) *For each $z \in [\hat{n}]$ the preference list of each agent $x \in \{\alpha_i^j[z], \beta_j^i[z]\}$ satisfies $\{\alpha_i^j[z], \beta_j^i[z], \gamma_i^j[z-1]\} \succ_x \{\alpha_i^j[z], \beta_j^i[z], \gamma_i^j[z]\} \succ_x \dots$.*
- (vi) *For each $z \in [\hat{n}-1]$ and each triple $B \neq \{\alpha_i^j[z+1], \beta_j^i[z+1], \gamma_i^j[z]\}$ with $B \succeq_{\gamma_i^j[z]} \{\alpha_i^j[z+1], \beta_j^i[z+1], \gamma_i^j[z]\}$ it holds that $B \cap \{\alpha_i^j[z], \beta_j^i[z]\} \neq \emptyset$.*
- (vii) *For each $B \neq \{w_j^i, w_j^p, w_j^q\}$ with $B \succeq_{w_j^i} \{w_j^i, w_j^p, w_j^q\}$ we have $B \cap \{\alpha_i^j[\hat{n}], \beta_j^i[\hat{n}]\} \neq \emptyset$.*

Finally, we show the correctness, i.e., “ $I = (X, \mathcal{S})$ admits an exact cover if and only if the constructed instance admits a 3-stable matching” via the following lemmas. Lemma 6 shows the “only if” direction and Lemma 8 the other.

► **Lemma 6** (\star). *If $\mathcal{K} \subset \mathcal{S}$ is an exact cover of I , then the following 3-matching Π is stable.*

- *For each $S_j \in \mathcal{K}$ with $S_j = \{i, p, q\}$ add $\{w_j^i, w_j^p, w_j^q\}$ to Π .*
- *For each element $i \in X$ and each set $S_j \in \mathcal{S}$ with $i \in S_j$, call the agents in the associated star-gadget along with the tail agents $0, \dots, 11, h_j^i, f_j^i$, and g_j^i .*
 - *Add $H_j^i, \{5, 10, 11\}, \{1, 6, 8\}, \{2, 3, 7\}$, and $\{0, 4, 9\}$ to Π .*
 - *If $S_j \in \mathcal{K}$, then add all triples $\{\alpha_i^j[z], \beta_j^i[z], \gamma_i^j[z-1]\}$, $z \in [\hat{n}]$, to Π . Otherwise, add all triples $A_i^j[z]$, $z \in [\hat{n}]$, to Π .*
- *For each element $i \in X$ let S_k, S_r be the two sets which contain i , but are not chosen in the exact cover \mathcal{K} . Add $\{u_i, u_i^k, u_i^r\}$ to Π .*

The proof of the other direction is based on the following properties.

► **Lemma 7** (\star). *Let Π be a 3-stable matching of the constructed instance. For each element $i \in X$ and each set S_j with $S_j = \{i, p, q\}$, the following holds:*

- (i) $H_j^i \in \Pi$.
- (ii) Π contains either all triples $\{\alpha_i^j[z], \beta_j^i[z], \gamma_i^j[z]\}$ or all triples $\{\alpha_i^j[z], \beta_j^i[z], \gamma_i^j[z-1]\}$, $z \in [\hat{n}]$.

Now, we consider the “if” direction.

► **Lemma 8.** *If Π is a 3-stable matching, then the subcollection \mathcal{K} with $\mathcal{K} = \{S_j \in \mathcal{S} \mid \{\alpha_i^j[1], \beta_i^j[1], \gamma_i^j[0]\} \in \Pi \text{ for some } i \in S_j\}$ is an exact cover.*

Proof. First of all, for each two chosen $S_j, S_k \in \mathcal{K}$ we observe that it cannot happen that $S_j \cap S_k \neq \emptyset$ as otherwise $\{u_i, u_i^j, u_i^k\}$ is a blocking triple; recall that $\gamma_i^j[0] = u_i^j$ and $\gamma_i^k[0] = u_i^k$. It remains to show that \mathcal{K} covers each element at least once.

Now, for each element $i \in X$, let S_j, S_k, S_r denote the three sets that contain i . We claim that at least one of S_j, S_k, S_r belongs to \mathcal{K} because of the following. If $S_j \notin \mathcal{K}$, then by construction, it follows that $T = \{\alpha_i^j[1], \beta_i^j[1], \gamma_i^j[0]\} \notin \Pi$. By Lemma 7(ii), it follows that $A_i^j[1] \in \Pi$. Since T is the most-preferred triple of both $\alpha_i^j[1]$ and $\beta_i^j[1]$ (see Observation 5(v)), by stability, u_i^j must be matched in a triple which she weakly prefers to T . Since $A_i^j[1] \in \Pi$, it follows that either $\{u_i^j, u_i^k, u_i^r\} \in \Pi$ or $\{u_i^j, u_i, v\} \in \Pi$ for some $v \in \{u_i^k, u_i^r\}$. It cannot happen that $\{u_i^j, u_i^k, u_i^r\} \in \Pi$ as otherwise there will be at least three blocking triples, including $\{u_i, u_i^j, u_i^r\}$. Hence, $\{u_i^j, u_i, v\} \in \Pi$ for some $v \in \{u_i^k, u_i^r\}$. Without loss of generality, assume that $v = u_i^k$. Then, it is straightforward to check that $\{u_i^r, \alpha_i^r[1], \beta_i^r[1]\} \in \Pi$. This implies that $S_r \in \mathcal{K}$.

To complete the correctness proof, we show that for each element $p \in S_r \setminus \{i\}$ it holds that $\{\alpha_p^r[1], \beta_p^r[1], \gamma_p^r[0]\} \in \Pi$. Let $S_r = \{i, p, q\}$. Since $S_r \in \mathcal{K}$, by definition and by Lemma 7(ii), we infer that $\{\alpha_i^r[\hat{n}], \beta_i^r[\hat{n}], \gamma_i^r[\hat{n}-1]\} \in \Pi$ (for some constant \hat{n} defined in the construction). We infer that $\{w_r^i, w_r^p, w_r^q\} \in \Pi$ due to the following: By Lemma 7(i), we know that $H_r^i \in \Pi$; recall that $H_r^i = \{f_r^i, g_r^i, h_r^i\}$. Since both f_r^i and g_r^i prefer $\{f_r^i, g_r^i, w_r^i\}$ to H_r^i (see the first part of Observation 5(iii)), it follows by stability that $\Pi(w_r^i) \succeq_{w_r^i} \{f_r^i, g_r^i, w_r^i\}$. By Observation 5(vii), we infer that $\Pi(w_r^i) = \{w_r^i, w_r^p, w_r^q\}$ since $\alpha_i^r[\hat{n}]$ and $\beta_i^r[\hat{n}]$ are not available anymore. This means that $A_p^r[\hat{n}'], A_q^r[\hat{n}''] \notin \Pi$ since $w_r^p = \gamma_p^r[\hat{n}']$ and $w_r^q = \gamma_q^r[\hat{n}'']$ (for some constants \hat{n}' and \hat{n}''). Consequently, we infer by Lemma 7(ii) that $\{\alpha_p^r[1], \beta_p^r[1], \gamma_p^r[0]\}, \{\alpha_q^r[1], \beta_q^r[1], \gamma_q^r[0]\} \in \Pi$, as desired. ◀

This concludes the proof of Theorem 1 for $d = 3$.

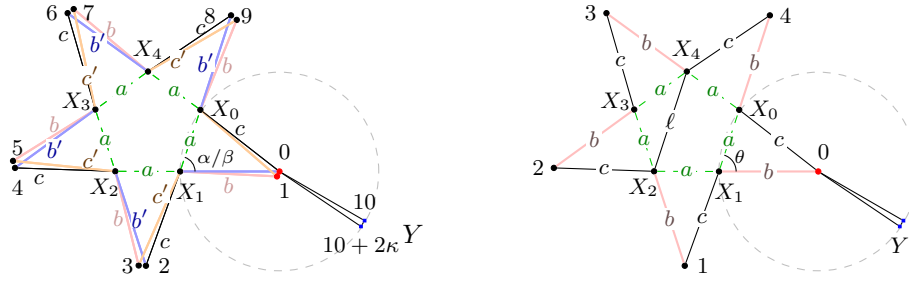
4 EUCLID-d-SR with $d \geq 4$

In this section we look at the cases where $d \geq 4$, and let $\kappa := \lfloor (d-1)/2 \rfloor$. The general idea of the reduction is similar to the case where $d = 3$, and we still reduce from PC-X3C. Briefly put, we adapt the star-gadget from Example 2. However, depending on whether d is even or not, we need to carefully revise the star-gadget from Example 2 to make sure the enforcement gadget works. We will replace each pentagon-agent with a subset of agents of size κ , and each further agent from the triangle with two agents if d is even. We also need to update both the replacement and the enforcement gadget. In Subsection 4.1, we describe in detail what the new star-gadgets and the the remaining gadgets look like, and how they are connected to each other. In Subsection 4.2 we show the correctness.

4.1 The construction

We first describe the adapted star-gadgets through the following example (also see Figure 4).

► **Example 9.** We first consider the construction for even d , i.e., $d = 2\kappa + 2$. Consider an instance with $7\kappa + 11$ agents called W where 5κ agents are embedded as the five vertices of a pentagon with κ agents at each vertex of the pentagon. We denote the five sets of points at the five vertices of the pentagon as X_0, \dots, X_4 . All points in each cluster $X_i, 0 \leq i \leq 4$,



■ **Figure 4** A star-structured instance adapted from Arkin et al. [1], similar to Example 2. The left one is for even d , while the right one is for odd d , both described in Example 9. See the caption of Example 2 for further explanation regarding the colors of the edges.

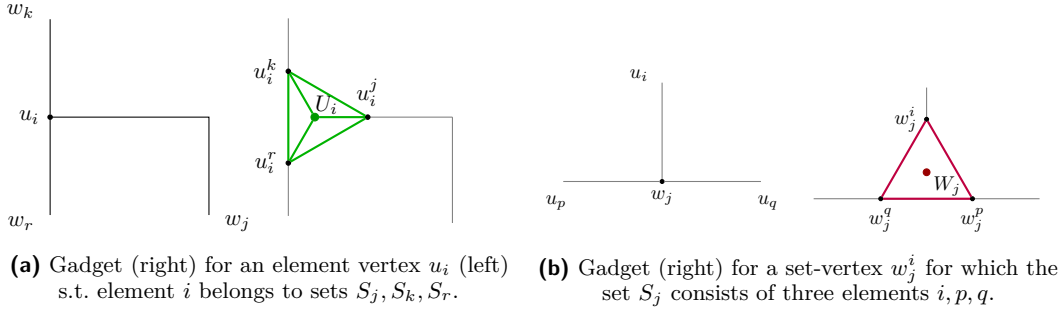
are embedded within an enclosing circle of radius close to zero, with the intention that a d -matching is stable only if all agents in X_i are matched together. For each $i \in \{0, \dots, 4\}$, the distance between each point X_i and each point in $X_{i+1 \bmod 5}$ is in the range of $[a, a + \varepsilon_d]$, while the distance between each point in X_i and each point in $X_{i+2 \bmod 5}$ is in the range of $[\ell, \ell + \varepsilon_d]$. There are 10 points $\{0, \dots, 9\}$ that form a star with the pentagon, as shown in Figure 4 (left). For each $i \in \{0, \dots, 4\}$, embed the points $2i$ and $2i + 1$ as follows: point $2i$ is at a distance between c and $c + \varepsilon_d$ to every point in X_i , and at a distance between b' and $b' + \varepsilon_d$ to every point in $X_{i+1 \bmod 5}$. Point $2i + 1$ is at a distance between c' and $c' + \varepsilon_d$ to every point in X_i , and at a distance between b and $b + \varepsilon_d$ to every point in $X_{i+1 \bmod 5}$. Finally, the distance $\delta(2i, 2i + 1)$ is close to 0. Here the mentioned values satisfy the following relations $a < b < c < \ell$, $b < b' < \ell$, $c < c' < \ell$, $b + b' < 3a$, $c + c' < 3a$, $b + b' < a + \ell$, and $c + c' < b + \ell$.

The remaining $2\kappa + 1$ points, denoted by $10, \dots, 10 + 2\kappa$ in the figure, are called Y ; note that $|Y| = 2\kappa + 1 = d - 1$. Together, $W := \bigcup_{i \in \{0, \dots, 4\}} X_i \cup \{0, \dots, 9\} \cup Y$. All points in Y are embedded within an enclosing ball with radius close to zero. For each point y in Y , it holds that $b - \varepsilon_d \leq \delta(0, y) < b$ and $b - \varepsilon_d \leq \delta(1, y) < b$, and for each each point w in $W \setminus (\{0, 1\} \cup Y)$ it holds that $\delta(w, y) > \ell$. Points 0 and 1 are the two points from $W \setminus Y$ which are closest to the points in Y .

To specify the embedding, We first fix points 0, 1, and Y such that the distances between them are as stated above and they are embedded roughly around a straight line. Then, we fix the positions of X_0, X_1 , and the centroid of the pentagon to ensure the values a, b, b', c, c' , and ℓ satisfy the above relations. For each $i \in \{0, 1, 2, 3, 4\}$ and each two points $x \in X_i$ and $x' \in X_{i+1 \bmod 5}$, the angle α (resp. β) at the points $2i, x$, and x' (resp. $2i + 1, x'$, and x) is less than 90 degrees. The angle at points y, j , and x ($y \in Y, \{i, j\} = \{0, 1\}, x \in X_i$) is more than 90 degrees. After fixing $X_0, X_1, 0$, and 1, we can determine the other points by simple calculations.

Now, we turn to odd d , i.e., $d = 2\kappa + 1$. Instead of having ten points $\{0, \dots, 9\}$, we create five points that form a star with the pentagon. Consider an instance with $7\kappa + 5$ agents called W where 5κ agents are embedded to replace the five vertices of a pentagon with κ agents at each vertex of the pentagon. That is, each vertex of the pentagon is a cluster of points. note the five clusters of points by X_0, X_1, X_2, X_3 , and X_4 . There are five points $\{0, 1, 2, 3, 4\}$ that form a star with the pentagon, as in Example 2 (see Figure 4 (right)). Point i is at a distance b from X_i and c from $X_{i+1 \bmod 5}$, for each $i \in \{0, \dots, 4\}$ where $a < b < c < \ell$ and $b < 2a$.

36:12 Multi-Dimensional Stable Roommates in 2-Dimensional Euclidean Space



■ **Figure 5** Element and set gadgets described in Subsection 4.1.

The remaining 2κ points are called Y . Together, $W := \bigcup_{i \in \{0, \dots, 4\}} X_i \cup \{0, 1, 2, 3, 4\} \cup Y$.

All points in Y are embedded within an enclosing circle with radius close to zero. For each point y in Y , it holds that $b - \varepsilon \leq \delta(0, y) < b$, and for each each point w in $W \setminus (\{0\} \cup Y)$ it holds that $\delta(y, w) > \ell$. Point 0 is the only point from $W \setminus Y$ which is closest to the points in Y . The remaining unmentioned points are at distance at least $b/2$ to the points Y . We specify the embeddings of the agents similarly to the one for even d .

Using a similar reasoning as to Example 2, we claim that the above embeddings are feasible.

Since the distance between each two points in X_i is close to zero, we assume it to be 0 for ease of reasoning. The following lemma summarizes the crucial effect of the star-gadget.

► **Lemma 10** (\star). *Every d -stable matching Π of the instance in Example 9 satisfy the following.*

- If d is even, then $\Pi(0) \cap Y \neq \emptyset$ or $\Pi(1) \cap Y \neq \emptyset$.
- If d is odd, then $\Pi(0) = Y \cup \{0\}$.

The remaining gadgets. Let $I = (X, \mathcal{S})$ be an instance of PC-X3C. Similarly to the case with $d = 3$, we first embed the associated graph $G(I) = (U \uplus W, E)$ into a 2-dimensional grid with edges drawn using line segments of length at least $L \geq 200$, and with parallel lines at least $4L$ grid squares apart. The element- and the edge-gadget are almost the same as the ones describe in Subsection 3.1. The only difference is that we replace each element-agent u_i (for $u_i \in U$) with a size- $(d - 2)$ coalition U_i that are embedded so close to each other that any stable matching must match them together. Similarly, for each $z \in [\hat{n}]$ (recall that \hat{n} is a constant as defined in the Subsection 3.1) and $w_j \in W$, we replace the two agents $\alpha_i^j[z]$ and $\beta_i^j[z]$ with a size- $(d - 1)$ coalition $\hat{A}_i^j[z]$ such that the distance between each pair of points in $\hat{A}_i^j[z]$ is close to zero, and define $A_i^j[z] := \hat{A}_i^j[z] \cup \{\gamma_i^j[z]\}$. For each set-vertex $w_j \in W$, assume that the three connecting edges in $G(I)$ are going rightward, leftward, and upward, connecting the element-vertices u_i, u_p, u_q , respectively. We create three set-agents, called w_j^i, w_j^p, w_j^q , and an additional coalition W_j of size $d - 3$ and as before, define $w_j^i = \gamma_j^i[\hat{n}]$. We embed them into \mathbb{R}^2 in such a way that w_j^i, w_j^p, w_j^q are on the segment of the rightward, leftward, and upward edge, respectively, and are of equidistance 17.5 to each other, and the coalition W_j is embedded in the center so that the distance between any two of them is close to zero. Moreover, the largest distance from any agent of W_j to any agent of $\{w_j^i, w_j^p, w_j^q\}$ is 10. See Figure 5 for an illustration.

We remark that by the construction of the set-gadget and the edge-gadget, each set-agent w_j^i prefers coalition $A_i^j[\hat{n}]$ (recall that $\gamma_i^j[z] = w_j^i$) to coalition $\{w_j^i, w_j^p, w_j^q\} \cup W_j$ since the sum of distances from w_j^i to the latter coalition is $17.5 + 17.5 + 10(d - 3) > (d - 1) \cdot (10 - \varepsilon)$.

To ensure that one of the two coalitions is chosen, we make use of the star-gadgets from Example 9. Define $b := 22.6$ and $c := 22.7$. We create an agent-subset F_j^i of size $d - 1$ and agent h_j^i and a star-gadget W as described in Example 9, with Y being the extra $d - 1$ agents such that the most preferred coalition of each agent in Y is $Y \cup \{h_j^i\}$. Note that F_j^i has the same role as $\{f_j^i, g_j^i\}$ in the case for $d = 3$.

- The distance between each two agents in F_j^i is close to zero.
- The distance from each agent in F_j^i to each agent in $\hat{A}_i^j[\hat{n}]$ is in the range of $[10 + \varepsilon, 10 + 2\varepsilon)$.
- The distance from each agent in F_j^i to agent w_j^i is $10 + \frac{15}{d-1}$.
- The distance from each agent in F_j^i to agent h_j^i is $15 + 2\varepsilon$.
- The distance from agent h_j^i and each agent Y is $15 + 3\varepsilon$.
- The distance from each agent Y to 0 (and also to 1 if d is even) is $15 + 4\varepsilon$.

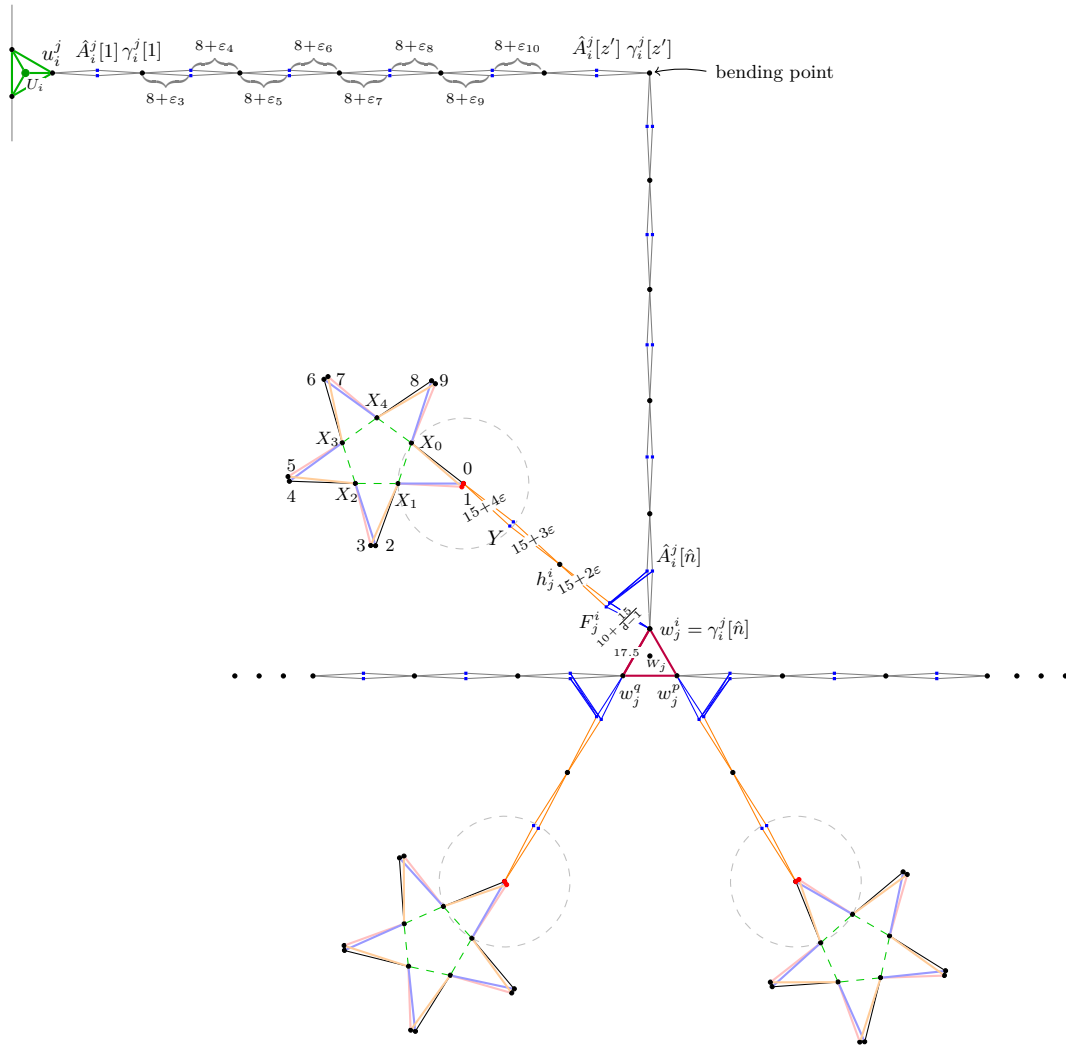
Finally, we create two types of garbage collector agents to match with some left over agents. For each added star gadget corresponding to S_j and $i \in S_j$, we create $O(\kappa)$ garbage collector agents R_j^i as follows: If d is odd, set $|R_j^i| := d - \kappa - 2$. Otherwise if $d \leq 6$, set $|R_j^i| := 2d - \kappa - 5$, and otherwise set $|R_j^i| := d - \kappa - 5$. These agents have distance close to zero to each other. For each $y \in R_j^i$ it holds that $\ell < \delta(y, x) < 2\ell < \delta(y, x')$, where x is an agent from the same star and x' is an agent from neither R_j^i or the same star. It is straightforward to see that the distance between any two agents from different star-gadgets is larger than ℓ , and the distance from an agent in W to an agent to a set-gadget is at larger ℓ , where a, b, b', c' , and ℓ are as defined in Example 9. Lastly, we add $m - n$ triples of additional garbage collector agents. The agents in each triple have distance close to zero to each other but is far away from the other agents. Note that each triple will be matched to some W_j whenever S_j is not chosen to the exact cover. See Figure 6 (for even d , without the garbage collector agents) for an illustration. This completes the description of the construction, which clearly can be done in polynomial time.

4.2 The correctness proof for $d \geq 4$

The reasoning for the correctness is similar to the one for $d = 3$. For the forward direction, assume that (X, \mathcal{S}) admits an exact cover \mathcal{K} . Then, using a reasoning similar to the one for $d = 3$, one can verify that the following d -matching Π is stable; recall that $\kappa = \lfloor (d - 1)/2 \rfloor$.

- For each $S_j \in \mathcal{K}$ with $S_j = \{i, p, q\}$ add $\{w_j^i, w_j^p, w_j^q\} \cup W_j$ to Π .
- For each element $i \in X$ let S_k, S_r be the two sets which contain i , but are not chosen in the exact cover \mathcal{K} . Add $U_i \cup \{u_i^k, u_i^r\}$ to Π . For each $S_j \notin \mathcal{K}$, take a triple of garbage collector agents (of the second type) and match them with W_j .
- For each element $i \in X$ and each set $S_j \in \mathcal{S}$ with $i \in S_j$, call the agents in the associated star-gadget along with the tail $X_0 \cup \dots \cup X_4 \cup \{0, 1, 2, 3, 4, h_j^i\} \cup Y \cup \{F_j^i\} \cup \{5, 6, 7, 8, 9 \mid \text{if } d \text{ odd}\}$. If $S_j \in \mathcal{K}$, then add all $\hat{A}_i^j[z] \cup \{\gamma_j^i[z - 1]\}$, $z \in [\hat{n}]$, to Π . Otherwise, add all $A_i^j[z]$, $z \in [\hat{n}]$, to Π . Add $F_j^i \cup \{h_j^i\}$ and $Y \cup \{0\}$ to Π . If d is odd, add $X_1 \cup X_2 \cup \{1\}$ and $X_3 \cup X_4 \cup \{3\}$ to Π . Otherwise, add $X_1 \cup X_2 \cup \{2, 3\}$ and $X_3 \cup X_4 \cup \{6, 7\}$ to Π . Next, if $d \leq 6$, then match X_0 with $d - \kappa$ agents from $(1, 8, 9, 4)$ (in this sequence) to Π . In any case, match the remaining star-agents with R_j^i .

The proof for the backward direction works analogously to $d = 3$ and is deferred to the full version [8].



■ **Figure 6** Gadget for edge $\{u_i, w_j\}$ in $G(I)$ with $S_j = \{i, p, q\}$ for the case when d is even, omitting the garbage collector agents for the sake of brevity.

5 Conclusion and Outlook

Establishing the first complexity results in the study of multi-dimensional stable matchings for Euclidean preferences, we show that d -SR remains NP-hard for Euclidean preferences and for all fixed $d \geq 3$. The gadgets in the reductions may be useful for other matching and hedonic games problems with Euclidean preferences.

Our Euclidean preference model assumes that the preferences over coalitions are based on the sum of distances to all individual agents in the coalition. It would be interesting to see whether taking the maximum or the minimum distance to the coalition members instead of the sum would change the complexity. Furthermore, it would be interesting to see whether restricting the agents' embedding to 1-dimensional Euclidean space could lower the complexity. We were not able to identify the complexity for this restricted variant, but conjecture that it can be solved in polynomial time. Note that in 1-dimensional Euclidean space, a 3-stable matching for the maximum distance setting always exists, which can be found by greedily finding three consecutive agents which are closest to each other and matching them.



References

- 1 Esther M Arkin, Sang Won Bae, Alon Efrat, Kazuya Okamoto, Joseph SB Mitchell, and Valentin Polishchuk. Geometric stable roommates. *Information Processing Letters*, 109(4):219–224, 2009.
- 2 Giuseppe Di Battista, Giuseppe Liotta, and Francesco Vargiu. Spirality and optimal orthogonal drawings. *SIAM Journal on Computing*, 27(6):1764–1811, 1998.
- 3 Vittorio Bilò, Gianpiero Monaco, and Luca Moscardelli. Hedonic games with fixed-size coalitions. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI '22)*, 2022. To appear.
- 4 Péter Biró and Eric McDermid. Three-sided stable matchings with cyclic preferences. *Algorithmica*, 58(1):5–18, 2010.
- 5 Robert Bredereck, Jiehua Chen, Ugo P. Finnendahl, and Rolf Niedermeier. Stable roommate with narcissistic, single-peaked, and single-crossing preferences. *Autonomous Agents and Multi-Agent Systems*, 34(53):1–29, 2020.
- 6 Robert Bredereck, Klaus Heeger, Dusan Knop, and Rolf Niedermeier. Multidimensional stable roommates with master list. In *Proceedings of the 16th International Conference on Web and Internet Economics (WINE '20)*, volume 12495 of *LNCS*, pages 59–73, 2020.
- 7 Jiehua Chen, Gergely Csáji, Sanjukta Roy, and Sofia Simola. Cores in friend-oriented hedonic games: Verification is surprisingly harder than searching. Technical report, arXiv, 2022. [arXiv:2203.09655](https://arxiv.org/abs/2203.09655).
- 8 Jiehua Chen and Sanjukta Roy. Multi-dimensional stable roommates in 2-dimensional Euclidean space. Technical report, arXiv, 2022. [arXiv:2108.03868](https://arxiv.org/abs/2108.03868).
- 9 Ágnes Cseh, Tamás Fleiner, and Petra Harján. Pareto optimal coalitions of fixed size. *Journal of Mechanism and Institution Design*, 4(1):87–108, 2019.
- 10 Vladimir G. Deineko and Gerhard J. Woeginger. Two hardness results for core stability in hedonic coalition formation games. *Discrete Applied Mathematics*, 161(13-14):1837–1842, 2013.
- 11 Dinko Dimitrov, Peter Borm, Ruud Hendrickx, and Shao Chin Sung. Simple priorities and core stability in hedonic games. *Social Choice and Welfare*, 26:421–433, 2006.
- 12 Jacques H. Dréze and Joseph Greenberg. Hedonic coalitions: Optimality and stability. *Econometrica*, 48(4):98–1003, 1980.
- 13 Kimmo Eriksson, Jonas Sjöstrand, and Pontus Strimling. Three-dimensional stable matching with cyclic preferences. *Mathematical Social Sciences*, 52(1):77–87, 2006.
- 14 David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 120(5):386–391, 1962.
- 15 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 16 Chien-Chung Huang. Two’s company, three’s a crowd: Stable family and threesome roommates problems. In *Proceedings of the 15th Annual European Symposium (ESA '07)*, volume 4698 of *LNCS*, pages 558–569, 2007.
- 17 Robert W. Irving. An efficient algorithm for the ‘stable roommates’ problem. *Journal of Algorithms*, 6(4):577–595, 1985.
- 18 Kazuo Iwama, Shuichi Miyazaki, and Kazuya Okamoto. Stable roommates problem with triple rooms. In *Proceedings of the 10th KOREA-JAPAN joint workshop on algorithms and computation (WAAC '07)*, pages 105–112, 2007.
- 19 Donald Ervin Knuth. *Mariages stables et leurs relations avec d’autres problèmes combinatoires: introduction à l’analyse mathématique des algorithmes*. Les Presses de l’Université de Montréal, 1976.
- 20 Chi-Kit Lam and C Gregory Plaxton. On the existence of three-dimensional stable matchings with cyclic preferences. In *Proceedings of the 12th International Symposium on Algorithmic Game Theory (SAGT '19)*, volume 11801 of *LNCS*, pages 329–342, 2019.


36:16 Multi-Dimensional Stable Roommates in 2-Dimensional Euclidean Space

- 21 David F. Manlove. *Algorithmics of Matching Under Preferences*, volume 2 of *Series on Theoretical Computer Science*. World Scientific, 2013.
- 22 Michael McKay and David F. Manlove. The three-dimensional stable roommates problem with additively separable preferences. In *Proceedings of the 14th International Symposium on Algorithmic Game Theory (SAGT '21)*, volume 12885 of *LNCS*, pages 266–280, 2021.
- 23 Cristopher Moore and J. M. Robson. Hard tiling problems with simple tiles. *Discrete & Computational Geometry*, 26(4):573–590, 2001.
- 24 Kazunori Ohta, Nathanaël Barrot, Anisse Ismaili, Yuko Sakurai, and Makoto Yokoo. Core stability in hedonic games among friends and enemies: Impact of neutrals. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI '17)*, pages 359–365, 2017.
- 25 Leslie G. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, C-30(2):13–140, 1981.
- 26 Gerhard J. Woeginger. Core stability in hedonic coalition formation. In *Proceedings of the 39th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '13)*, volume 7741 of *LNCS*, pages 33–50, 2013.
- 27 Mason Wright and Yevgeniy Vorobeychik. Mechanism design for team formation. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI '15)*, pages 1050–1056, 2015.

Spanner Approximations in Practice

Markus Chimani  

Theoretical Computer Science, Universität Osnabrück, Germany

Finn Stutzenstein 

Theoretical Computer Science, Universität Osnabrück, Germany

Abstract

A multiplicative α -spanner H is a subgraph of $G = (V, E)$ with the same vertices and fewer edges that preserves distances up to the factor α , i.e., $d_H(u, v) \leq \alpha \cdot d_G(u, v)$ for all vertices u, v . While many algorithms have been developed to find good spanners in terms of approximation guarantees, no experimental studies comparing different approaches exist. We implemented a rich selection of those algorithms and evaluate them on a variety of instances regarding, e.g., their running time, sparseness, lightness, and effective stretch.

2012 ACM Subject Classification Theory of computation \rightarrow Sparsification and spanners; General and reference \rightarrow Experimentation

Keywords and phrases Graph spanners, experimental study, algorithm engineering

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.37

Related Version *Extended Version*: <https://arxiv.org/abs/2107.02018>

1 Introduction

Consider a directed or undirected graph $G = (V, E)$ with n vertices and m edges. The distance $d_G(u, v)$ is the length of a shortest path between two vertices u and v , possibly subject to edge weights $w: E \rightarrow \mathbb{R}^+$. A *spanner* is a sparse subgraph of G that preserves these distances to some quality degree. Spanners were introduced by Peleg and Schäffer [38] after the first mention by Awerbuch [5]. There are many spanner variants, see Ahmed et al [1] for a survey. In this paper we are going to focus on the probably most popular variant:

► **Definition 1** (Multiplicative α -Spanner [38]). *Given a directed or undirected graph $G = (V, E)$ and a stretch $\alpha \geq 1$, a multiplicative α -spanner $H = (V, E')$ is a subgraph of G with $E' \subseteq E$ such that $d_H(u, v) \leq \alpha \cdot d_G(u, v)$ holds for all pairs $(u, v) \in V \times V$.*

For simplicity we use the term *spanner* for the multiplicative α -spanner in the following.

Given a graph G , finding any spanner is trivial since G is a spanner of itself. The problem we are interested in is to find a *good* spanner. Next to the stretch α itself, there are three basic measures of a spanner's quality: the *size* is the number of edges $|E'|$; the *sparseness* $s(H) = \frac{|E'|}{|E|}$ is the relative size w.r.t. the original graph, with a lower value indicating a sparser spanner; the *lightness* $\ell(H) = \frac{W(E')}{W(\text{MST}(G))}$ with $W(F) = \sum_{e \in F} w(e)$ compares the weight of the spanner to the weight a minimum spanning tree in G .

► **Definition 2** (Sparsest (Lightest) α -Spanner). *Given a graph $G = (V, E)$ and a stretch $\alpha \geq 1$, find a sparsest (lightest) multiplicative α -spanner $H = (V, E')$ such that $s(H) \leq s(H')$ ($\ell(H) \leq \ell(H')$, respectively) for all other spanners H' of G .*

Finding good spanners is motivated by several applications, e.g., ranging from routing problems [5, 40, 17, 43] to distributed computing [39, 18]. Also many theoretical applications require spanners, e.g., approximate distance oracles [37, 45], almost shortest paths [25, 22], or access control hierarchies [31, 9]. Thus, while finding a sparsest or lightest spanner is NP-hard [38, 11], a multitude of algorithms was developed to find sparse or light spanners.



© Markus Chimani and Finn Stutzenstein;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, 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

We are interested in the computation of spanners in practice. Farshi and Gudmundsson [27] compared specialized algorithms based on WSPD-graphs [12] and Θ -graphs [16, 32] for the geometric case; Bouts et al. [10] presented experiments, e.g., on multilevel subset additive spanners. Regarding exact algorithms, Sigurd and Zachariassen [44] developed in 2004 an exact spanner algorithm based on an ILP with an exponential number of variables, each representing a shortest path. Graphs with up to 64 vertices were tested but not all instances could be solved within the time limit of 30 minutes. Ahmed et al. [2] published a compact ILP based on multicommodity flow in 2019, which can be solved by standard B&B-techniques. They tested graphs with up to 100 vertices, but the solver needed up to 40 hours to solve the largest instances on a high-performance 400-node cluster. They did not compare the results to [44] or other popular heuristics. Apart from these studies, there seem to be no further experimental studies, in particular none for the arguably most prominent base case of multiplicative α -spanners in non-geometric settings.

Contribution. Despite the multitude of applications, there is an obvious total lack of practical experience with polynomial algorithms for non-geometric spanners. While there are various intrinsically different algorithmic ideas known in literature, there is no understanding which of these concepts lend themselves to fruitful practical applications. The main focus of this paper is thus to gain some insight in the practical consequences of these algorithmically diverse ideas. – In Sect. 2, we provide an overview of the known algorithms. We argue our selection of algorithms and give implementation details in Sect. 3. In Sect. 4, we present the results of our experimental study.

2 Known Algorithms

Much work and effort has been put into developing different approaches to find spanners. We will categorize the existing literature into three main categories: greedy algorithms, approximation algorithms, and other methods. For a rich theoretical survey of graph spanners, their theoretical background and algorithms, see Ahmed et al. [1].

Greedy algorithms. One of the earliest algorithms is by Althöfer et al. [4] and is reminiscent of Kruskal’s algorithm. The algorithm ADDJS, often named *Basic Greedy Spanner Algorithm*, starts with a spanner without edges and adds edges $\{u, v\}$ with increasing weight to the spanner, if the shortest path between u and v is currently too long, i.e., $d_H(u, v) > \alpha \cdot w(\{u, v\})$. For a stretch $\alpha = 2k - 1$ ($k \in \mathbb{N}_{\geq 1}$), this algorithm creates a spanner of size $\mathcal{O}(n^{1+1/k})$ and lightness $\mathcal{O}(n/k)$. The running time is mostly dominated by a Dijkstra run for each edge: $\mathcal{O}(m(n^{1+1/k} + n \log n))$. – Roditty and Zwick [42] sped up to the distance calculation in the spanner by incrementally maintaining a shortest-path tree, but lose any lightness guarantee in the process. Thorup and Zwick [45] explored *approximate distance oracles*, yielding a spanner algorithm as a byproduct, but as before, no lightness guarantee can be given.

To further improve the quality measures, more refined analyses using ε -parameterized algorithms were introduced. Thereby, one allows the spanner to violate the stretch requirement by a factor of up to $1 + \varepsilon$, while the approximation ratio of the sparsity and lightness is measured against the best spanner not violating this requirement. Chandra et al. [13], Elkin and Solomon [26] and Elkin et al. [24] use various techniques (e.g., dynamic shortest-path trees, further auxiliary graphs, and distance oracles) and refined analyses to create a spanner with guarantees for size and lightness as well as maintaining a low running time. Alstrup et al. [3] provide the currently best guarantees; their algorithm creates a $((2k - 1)(1 + \varepsilon))$ -spanner

with size $\mathcal{O}_\varepsilon(n^{1+1/k})$ and lightness $\mathcal{O}_\varepsilon(n^{1/k})$ within a running time of $\mathcal{O}(n^{2+1/k+\varepsilon'})$. Thereby, \mathcal{O}_ε ignores polynomial factors depending on $1/\varepsilon$. However, ADDJS still has the best lightness guarantee while strictly providing an $\alpha = 2k - 1$ stretch and is existentially optimal, see Ahmed et al. [1, Chapter 4] for a more thorough theoretical insight. Additionally, most above modifications lead to very complex algorithms that are often not practically implementable due to the intricate additionally required data structures mentioned above.

Clustering algorithms. Baswana and Sen [6] (BS) gave a method based on growing clusters around sampled vertices. It is special compared to all other algorithms as it does not do any local or global distance calculation. It computes an $\alpha = 2k - 1$ spanner for a weighted, undirected graph in two phases. First, there are $k - 1$ rounds of growing clusters by sampling vertices and successively adding nearest neighbors to each cluster. Then, clusters are joined by adding all non-clustered vertices to the nearest adjacent cluster and interconnecting all clusters. This yields a spanner of size $\mathcal{O}(kn^{1+1/k})$ in an expected running time of $\mathcal{O}(km)$, but no lightness guarantee. For an unweighted graph, the size is bounded by $\mathcal{O}(n^{1+1/k} + kn)$.

Approximation algorithms. An early $\mathcal{O}(\log(m/n))$ -approximation in terms of size by Kortsarz and Peleg [34] was published in 1994 for the special case of a 2-spanner for unweighted and undirected graphs (algorithm KP). An edge e is *covered*, if it is not part of the spanner, but part of a triangle in the original graph and both other edges are in the spanner, so the spanner property holds for e . The idea is to cover a large number of edges while not adding too many edges to the spanner by finding dense subsets of vertices and adding connecting edges to the spanner. KP calculates *dense subsets* U_v of neighboring vertices $N(v)$ for each vertex v and continues with the densest subset, say U_w . The star $\{\{u, w\} | u \in U_w\}$ is added to the spanner. The edges within the dense subset are now covered and finding the densest subgraph allows for covering the most edges. This is repeated while there exists a dense subset with a density larger than 1. The running time is bounded by $\mathcal{O}(nm \cdot \text{MDS}(n, m))$ where $\text{MDS}(n, m)$ is the running time of the algorithm to solve the maximum density subgraph problem. Using Goldberg's algorithm [30], the resulting time complexity is $\mathcal{O}(m^2 n^3 \log n)$. There exist more advanced algorithms for the maximum density problems so the theoretical running time can be lowered to $\mathcal{O}(m^2 n^2 \log(n^2/m))$ [28].

For undirected graphs we know of no explicit further approximation results. One reason may be that ADDJS already has an $\mathcal{O}(n^{1+1/k})$ guarantee for the spanner size. For a connected original graph, a spanner must have at least $n - 1$ edges, as it has to be connected as well. This lower bound for an optimal solution thus results in a straight-forward $\mathcal{O}(n^{1/k})$ -approximation in terms of size for every undirected graph.

For directed graphs, Elkin and Peleg [25] gave the first $\tilde{\mathcal{O}}(n^{2/3})$ -approximation in the special case $\alpha = 3$. For general $\alpha \geq 3$, Bhattacharyya et al. [9] published an $\tilde{\mathcal{O}}(n^{1-1/\alpha})$ -approximation combining two techniques: The spanner is the union of the arc sets of two graphs. The first graph is obtained by solving an LP relaxation and rounding the solution. The second one is created by sampling vertices and growing BFS arborescences from them. Dinitz and Krauthgamer [20] improved the techniques and Berman et al. [8] gave the currently best solution for general stretch $\alpha \geq 1$ (algorithm BBMR): They introduced an $\mathcal{O}(n^{1/2} \log n)$ -approximation for weighted, directed graphs. For the special case $\alpha = 3$ (later extended to $\alpha = 4$ by Dinitz and Zhang [21]) a slightly modified approach can be taken to achieve an $\tilde{\mathcal{O}}(n^{1/3})$ -approximation for directed and unweighted graphs. The techniques used are similar to the approach of Bhattacharyya et al. [9]: The arcs of the original graphs are categorized as either *thin* or *thick* by the number of shortest paths connecting the endpoints of the

■ **Table 1** Considered algorithms. The letters G, A, C and P in the second column stand for greedy, approximation, clustering, and probabilistic, respectively. *Assume $\alpha = 2k - 1$, $k \in \mathbb{N}_{\geq 1}$.

Algorithm		$w(e)$	Stretch	Spanner Properties*	Running time
ADDJS [4]	G	✓	$\alpha \in \mathbb{R}_{\geq 1}$	$s(H) \in \mathcal{O}(n^{1+1/k})$ $\ell(H) \in \mathcal{O}(n/k)$ [see text]	$\mathcal{O}(m(n^{1+1/k} + n \log n))$
KP [34]	A	×	$\alpha = 2$	$\mathcal{O}(\log(m/n))$ -approx. in size	$\mathcal{O}(m^2 n^3 \log n)$
BBMRY [8]	A	✓	$\alpha \in \mathbb{R}_{> 1}$	$\mathcal{O}(n^{1/2} \log n)$ -approx. in size	poly
BS [6]	C	×	$\alpha \in \mathbb{N}_{\geq 3}$, odd	$s(H) \in \mathcal{O}(n^{1+1/k} + kn)$	expected $\mathcal{O}(km)$
BS [6]	C	✓	$\alpha \in \mathbb{N}_{\geq 3}$, odd	$s(H) \in \mathcal{O}(kn^{1+1/k})$	expected $\mathcal{O}(km)$
EN [23]	P	×	$\alpha \in \mathbb{N}_{\geq 1}$, odd	$s(H) \in \mathcal{O}(n^{1+1/k}/\varepsilon)$ $0 < \varepsilon < 1$	expected $\mathcal{O}(m)$, success prob. $\geq 1 - \varepsilon$

arc. An LP rounding covers all thin arcs, and growing arborescences from sampled vertices cover all thick arcs. For the (I)LP, *antispanners* were introduced. Given an arc (u, v) , an antispanner is an arc set A , such that no shortest path with a distance less than $\alpha \cdot d_G(u, v)$ exists between u and v in $G \setminus A$. A is a minimal antispanner for (u, v) if no $A' \subset A$ is an antispanner for (u, v) . The ILP tries to cover arcs so that at least one arc of each minimal antispanner is used. If this is achieved, the selected arcs resemble a feasible solution. Due to the exponential number of minimal antispanners, we require a separation oracle to solve the LP relaxation of this ILP with cutting planes. The separation oracle has an exponentially small probability in n of failing.

Randomized Algorithms. A probabilistic method was presented by Elkin and Neiman [23] to calculate a spanner with stretch $2k - 1$ for undirected, unweighted graphs (algorithm EN). They built upon Miller et al. [36, 35] and improved the size to $\mathcal{O}(n^{1+1/k}/\varepsilon)$. For each vertex u , a random value r_u is drawn from an exponential distribution. Each vertex broadcasts a message with r_u as the content to all vertices within distance k . A vertex x , receiving messages from some vertices $U \subset V$, stores $m_u(x) = r_u - d_G(u, x)$ for each vertex $u \in U$ together with an edge adjacent to x that lies on a shortest path between u and x . The edges belonging to the messages with $m_u(x) \geq \max_{u \in V} \{m_u(x)\} - 1$ are added to the spanner. Since the algorithm has a success probability of at least $1 - \varepsilon$, we can obtain an expected running time of $\mathcal{O}(m)$ by iterating.

3 Considered Algorithms and Implementation Details

As motivated in Sect. 1, we aim at covering very diverse algorithmic approaches and investigating their relative merits, rather than several algorithmic variants finetuning a common idea. Only once an understanding on the most worthwhile algorithmic concepts has been achieved, deeper comparisons of algorithmic variants would seem worthwhile.

Thus, we select five, intrinsically different, algorithmic approaches to analyze; Table 1 provides an overview of the characteristics of each algorithm. With the goal of small running times and larger instances, we decided against reestablishing the performance of the exact algorithms [44, 2]; as summarized in Sect. 1, their running times are nowhere competitive on the instances we aim to consider. As the representative for the fundamental greedy algorithms, we selected ADDJS. While the algorithms of Thorup and Zwick [45] and Roditty and Zwick [42] improve the running time, ADDJS promises better results w.r.t. size and lightness. As such, there was no good argument to consider these or the algorithmically

and implementationwise much more involved ε -parameterized variants at this stage of the scientific knowledge discovery (but see Sect. 5). This argument is further amplified by the fact that many of these variants require very sophisticated subalgorithms (see above) which are by themselves unknown territory from the algorithm engineering viewpoint.

From the approximation algorithms we selected KP and BBMRV as they are the state-of-the-art. For BBMRV, we implemented the variant for arbitrary α , not the special improvements for $\alpha = 3$ and $\alpha = 4$. In the following, we list some implementation specific details; all our implementations guarantee the running time of the algorithmic descriptions.

Althöfer et al. (ADDJS). We include a straightforward observation to speed-up the computation: We bound the shortest-path search for each pair of vertices by $\alpha \cdot w(\{u, v\})$ since finding a longer path (or no path at all) results in adding the edge $\{u, v\}$ to the spanner. Note that one is not limited to a stretch $\alpha \in \mathbb{N}_{\text{odd}}$; the algorithm works for arbitrary $\alpha \geq 1 \in \mathbb{R}$, but may only provide the size and lightness guarantees for the next odd integer.

Baswana and Sen (BS). We broadly follow the detailed algorithmic description of [6]. We changed only one small aspect: The second and third step of the first phase can be combined in a single pass. After sampling clusters, it is not necessary to first calculate the nearest neighboring sampled cluster for each vertex and store it in some data structure because the third step can directly follow and does not depend on other vertices.

Kortsarz and Peleg (KP). The algorithm mainly depends on an implementation for the *maximum density subgraph problem*. We use Goldberg's algorithm [30] due to its simplicity of implementation. It resembles a binary search with $\mathcal{O}(\log n)$ steps: A source and target vertex (s, t) are added to the graph and connected to all original vertices. In each step the weights of the new edges are chosen depending on the current search value. A minimum s - t -cut is calculated and based on the resulting cut it is decided, whether the lower or upper half of the search interval is used. To solve the minimum s - t -cut problem we use Goldberg-Tarjan's max-flow algorithm [29] with global relabeling and gap relabeling heuristic requiring $\mathcal{O}(n^2m)$ time. This results in a running time for Goldberg's maximum density subgraph algorithm of $\mathcal{O}(n^2m \log n)$ and, as described above, in a complete running time of $\mathcal{O}(m^2n^3 \log n)$ for KP.

The algorithm has to maintain three sets H^s , H^c , and H^u for the spanner edges, covered edges, and uncovered edges. In the implementation it is sufficient to only use the original graph's edge set $E = H^u$ (by shrinking G on the fly) and the spanner's edge set $E' = H^s$. There is no need to explicitly store covered edges H^c in some data structure. For each maximum dense subset U_w , the star edges are added to E' . The star edges and the edges of the implied dense subset graph $E(U_w) = \{\{u, v\} \in E \mid u, v \in U_w\}$, which are the covered edges, are removed from E , so that the covered edges are simply entirely removed. Since we have to calculate the density of the maximum density subset, we have $E(U_w)$ already available, so no extra work is required to calculate the edges induced by the subset of vertices.

Berman et al. (BBMRV). The original algorithmic description leaves some implementation details open. First, the property of an arc (s, t) being *thin* or *thick* has to be efficiently evaluated [8, Definition 2.2]. The definition uses a *local graph* $G^{s,t} = (V^{s,t}, E^{s,t})$ induced by all vertices belonging to paths from s to t with a length of at most $\alpha \cdot d_G(s, t)$. An arc (s, t) is thick, if $|V^{s,t}| \geq n^{1/2}$. Calculating all possible paths is not an option, so in- and out-arborescences can be used. First, we precalculate these arborescences since we need them for evaluating the local graphs and during sampling afterwards. For each vertex r , an in-

and out-arborescence rooted at r is calculated. The distances to other vertices x are saved in $d_{\text{in}}(r, x)$ ($d_{\text{out}}(r, x)$, respectively) together with the respective predecessor to be able to access the actual trees during sampling. To check if $|V^{s,t}| \geq n^{1/2}$ holds, we can use the equivalence: $x \in V^{s,t} \iff d_{\text{out}}(s, x) + d_{\text{in}}(t, x) \leq \alpha \cdot d_G(s, t)$. Checking each vertex x , we count $|V^{s,t}|$ and decide the property for each arc. This could be done by using either an in- or out-arborescence, but both types are needed for the sampling, so we can precalculate both anyhow. Consequently, the subsequent sampling does not involve any shortest-path calculations and can be done by looking up predecessor relations.

Next, given an arc (s, t) it must be checked, if a set $R \subseteq E$ settles the arc [8, Definition 2.3], i.e., R satisfies the α -spanner property for (s, t) . This is done by running a bounded Dijkstra on R with a running time of $\mathcal{O}(|R| + n \log n)$. The check whether an arc is settled is done quite often, e.g., during the separation method and in the minimization of an antispanner.

Finally, let us focus on the separation method and the creation of antispanners. The former is straightforward: After rounding the fractional solution, obtaining R , we check every thin arc, whether it is settled by R . For an unsettled thin arc (s, t) , we create an antispanner w.r.t. R (see [8, Claim 2.4]). Similar to the thin arc property where the vertices $V^{s,t}$ are required, we need the arc set $E^{s,t}$, as $A = E^{s,t} \setminus R$ is an antispanner. An arc (u, v) is an element of $E^{s,t}$ if and only if $d_{\text{out}}(s, u) + w(u, v) + d_{\text{in}}(t, v) \leq \alpha \cdot d_G(s, t)$; this check can reuse the precalculated arborescences. After finding the antispanner it has to be minimized. We greedily remove arcs from A , while the unsettled thin arc remains unsettled. If no further arc can be removed, A is a minimal antispanner.

Elkin and Neiman (EN). Provided the very compact description of the algorithm and no explicit pseudocode, one has to be very careful not to gloss over intricate important details. We implemented the algorithm in the standard centralized model. The authors also highlight the distributed and PRAM models (as the algorithm can be parallelized well), but we aim to keep it comparable to all other algorithms.

To start, remember that the edges saved together with the received messages must belong to *some* shortest u - x -path. Naturally, we store the edge where x received the message from. To ensure that the messages travel along shortest paths, we use a breadth-first-search (BFS) starting at u with a depth limit of k , without the need of a Dijkstra computation.

The algorithm includes two feasibility checks. If one fails, the algorithm does not provide a feasible solution. The first check, that $r_u < k$ holds for all $u \in V$, can be done directly while generating values r_u to let the algorithm fail quickly if the property does not hold. The second check at the end of the algorithm asserts that there are sufficiently many edges. Elkin and Neiman state that the spanner must have at least $n - 1$ edges and the algorithm fails, if $|E'| < n - 1$. This is only correct if the original graph is connected. If it has $c \geq 2$ components, we can use the condition $|E'| < n - c$ instead.

A major improvement in terms of memory consumption can be made when reversing the way the algorithm is formulated. The original formulation can chiefly be described as:

1. For each $u \in V$, broadcast messages $m_u(x)$ to every vertex x within distance k .
 2. For each receiving $x \in V$: Calculate $\max_{u \in V} \{m_u(x)\}$ and add edges to the spanner.
- Following this order, one has to save two $n \times n$ matrices containing the message values $m_u(x)$ and the edge where the messages are received. In a pilot study, this resulted in out-of-memory errors for very sparse graphs ($m \approx 2n$) with $n \approx 40,000$, which other algorithms can handle. To lower the memory consumption, we reversed the logic to focus on each receiving vertex x : A BFS starting at x identifies all vertices u that can broadcast a message to x . Then, the aggregation and edge addition can directly be done afterwards, before proceeding with the next x . To summarize all improvements, the code is provided in [15, Appendix A].

We note that it is possible to provide values for ε that are not in the interval $(0, 1)$. Especially $\varepsilon \geq 1$ can produce good spanners in practice with the downside of an increasing possibility to fail. For results of a pilot study to choose ε , see [15, Appendix B].

4 Experimental Results

We are now ready to present the main content of this paper. The considered algorithms are tested with a variety of instances and parameter settings, as described in the following. In Sect. 4.1 we take a look at the running time of each algorithm. We compare the quality of the resulting spanners in Sect. 4.2. Finally, in Sect. 4.3, we propose and briefly investigate two further graph measures we deem interesting in their own right: We ask about the mean degrees and the mean lengths of the shortest paths in weighted spanners (compared to the input graphs). Investigating these measures can help us to better understand the different behaviors and results of the various algorithms.

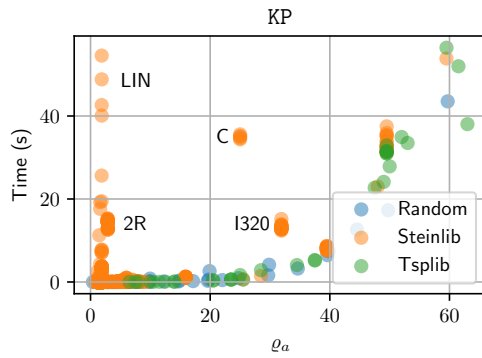
Setup. Since BBMR is the only algorithm that would work (with known quality guarantees) on directed graphs, we consider only undirected graphs. Edge weights are often a requirement in applications, but not all algorithms can take weighted graphs as inputs; thus we use each graph twice, once with and once without weights. In the following we will use the absolute density $\varrho_a(G) = m/n$ and the relative density $\varrho_r(G) = m/\binom{n}{2}$ to categorize graphs. In the experiments we consider integer stretches 2, 3, 4, 5, and 7. To test a variety of graph types, three instance libraries are used:

1. *Steinlib* [33]: This well-known library was originally created for the Steiner tree problem, but has also been used in many studies for related problems. Since several application areas of Steiner trees and spanners overlap, it seems judicious to consider this set. It contains 1207 graphs in 44 subsets. Over 62% of the graphs have $\varrho_r = m/\binom{n}{2} < 2.5\%$ (mostly sparse). On average, the graphs have 1189 vertices and 8549 edges; the largest graph has 38418 vertices.
2. *Tsplib* [41]: The well-established Tsplib contains complete weighted graphs. We omitted 27 of the 122 graphs as they have edge weights 0 or are directed. About half of the graphs have < 200 vertices; the largest graph has 7397 vertices. The graphs have 771 (250) vertices on average (median).
3. *Random*: We generated random Erdős-Rényi graphs with $|V| \in \{10, 20, 50, 100, 200, 500, 1000, 2000\}$, a relative density of $i/10$ for $i \in \{1, 2, \dots, 9\}$ and 10 graphs of each combination. Apart from the unweighted instances, we also consider two different random weight function: weights in the range $1 \pm 1/3$ (**W1**), and integer weights between 1 and n (**W2**). The former ensures that each edge is a shortest path between its end vertices, as is the case for unweighted instances, whereas the latter includes “obviously superfluous” edges. All graphs are available at [7].

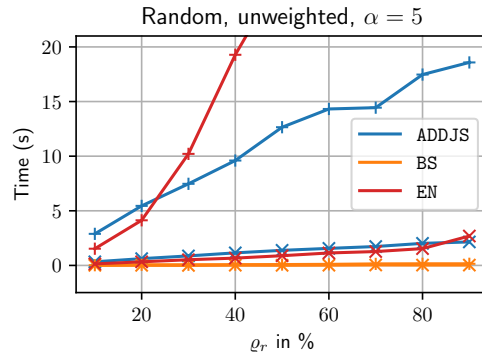
All implementations are freely available and will be part of the next release of the open source C++ *Open Graph algorithms and Datastructures Framework* [14] (www.ogdf.net). All experiments were performed on an Intel Xeon Gold 6134 with 256 GB RAM under Debian 10.2 using gcc 8.3.0-6 (-O3). For BBMR, we use CPLEX 20.1 [19] as the LP solver. When conducting an experimental study, and i.p. due to the page restriction, one has to decide between in-depth analyses of a smaller set of certain aspects or a broader approach of reporting the overall algorithmic behaviors over a larger field of questions and parameterizations. As our goal herein is to broadly assess competing algorithmic ideas, the second option makes more sense. However, all detailed data of our experiments is available at [7] for other researchers (or a subsequent longer journal version) to investigate further.

■ **Table 2** Solved instances per algorithm and average time for successful cases.

Library	ADDJS	BS	KP	BBMR	EN
Random	98.62% (2.96 s)	100.00% (0.03 s)	66.67% (2.46 s)	70.16% (3.07 s)	100.00% (3.10 s)
Steinlib	99.88% (0.48 s)	100.00% (0.11 s)	91.55% (1.91 s)	83.79% (5.32 s)	100.00% (0.37 s)
Tsplib	81.75% (4.98 s)	98.71% (1.47 s)	33.01% (14.75 s)	60.74% (5.70 s)	76.05% (3.11 s)
All	98.46% (0.92 s)	99.91% (0.21 s)	86.02% (2.30 s)	80.91% (5.21 s)	98.20% (0.66 s)



■ **Figure 1** Instances solved by KP within the time limit for each instance library. There are no further solved instances for larger ρ_a .



■ **Figure 2** Comparing ADDJS, BS, and EN for $n = 500$ (\times) and $n = 1000$ ($+$). BBMR cannot solve any of these instances.

Parameterizing the Randomized Algorithms. To fairly evaluate the randomized algorithms BS and EN, we first need to investigate their parameterizations. We report on these additional experiments in [15, Appendix B]. In the following, we use the most competitive settings of 1000 iterations for BS and 200 iterations for EN with $\varepsilon = 0.8$.

4.1 Running Time

We start with investigating the running times of our algorithms. We set a time limit of 60 seconds for each calculation: ADDJS, BS, and EN can solve the majority of all instances within this limit (cf. Table 2), and an additional pilot study shows that higher limits of 90 or 120 seconds do not significantly increase the number of solved instances. Before comparing the algorithms, we discuss their runtime behavior individually. In the following, when speaking about averages we consider the standard arithmetic mean, unless specified otherwise.

ADDJS. The running time is linear in m for fixed n . Almost all Steinlib instances can be solved within 6 seconds and all Tsplib instances with $n \leq 1200$ within the time limit. The densest instances over all libraries with $\rho_r \geq 90\%$ and $n \approx 1000$ can barely be solved within the time limit. The running time shows a dependency on the graph weights: unweighted graphs can be solved 33.45% faster than weighted ones. Furthermore, higher stretches are faster to compute. There is one peculiar outlier: while all other combinations of weights and stretches allow all Random instances to be solved within the time limit, $\alpha = 2$ for weighted (and **W1**-weighted, see also Sect. 4.2) instances is drastically slower and only achieves a success rate of 89%. This is also the only case where **W1** yields higher running times than **W2** and the unweighted case. Over the solved unweighted instances, $\alpha = 2$ is 42.61% slower than $\alpha = 3$ (while, e.g., $\alpha = 3$ is only 6.19% slower than $\alpha = 4$).

BS. The clustering approach is very fast. Every instance from Random is solved in under 0.6 seconds. The running time behaves linearly in m , as expected. Analogous to ADDJS, unweighted instances are faster, but in contrast, a *lower* stretch is faster as well. For Random, the running times for **W1** are in-between those of **W2** and unit weights. Almost all instances of the Tsplib can be solved; only the two largest instances ($n = 5915, 7397$) exceed the time limit for $\alpha \geq 5$. The majority of the Steinlib instances can be solved within a second. Generally, BS seems to have difficulties with large, very sparse graphs; for these graphs with $\varrho_r \leq 0.01\%$, the running time is up to 17.5 seconds (cf. [15, Appendix, Figure 8]). These are Steinlib instances from the subsets LIN, ALUT, and ALUE (grid graphs with holes, $n \geq 20000$).

KP. Surprisingly, there is no clear relationship between the running time and n , m , or ϱ_r ; however the absolute density is limiting: For instances with $40 \lesssim \varrho_a \lesssim 60$, the running time drastically starts to increase in comparison to $\varrho_a \lesssim 40$ and no instances with $\varrho_a \gtrsim 60$ can be solved in time (cf. Fig. 1). This behavior is consistent over all three libraries except for a handful of outliers from Tsplib. Regarding Steinlib, the sets 2R, I320, C, PUC and the largest instances of LIN require higher running times due to the very large n . The complete graphs of Tsplib can be solved with up to 120 vertices, which corresponds to $\varrho_a = 59.5$.

BBMR. Its running time depends on α and the weights: Similar to ADDJS, unweighted graphs are slightly faster (by 10.28%), but a higher stretch results in a lower running time. It also shares the performance degradation for $\alpha = 2$ on unweighted graphs. Overall, both the timeout rates and the running times are very high. Regarding the instance sizes, the number of edges has a disproportionate impact on the running time, i.e., only Steinlib instances with $m \leq 5000$ edges can be solved, regardless of the number of vertices. This is not surprising since the antispanner creation and minimization depends on the original graph's edges. Graphs from Random and Tsplib can only be solved if they have less than 250 nodes. Even though BBMR has a probability of failing, we never observed any failure during the experiments.

EN. We observe a linear running time in m . We measured the time until the first success, so previous failures are included in the running times. EN can solve graphs with up to 1000 vertices for Random and Tsplib, with a smaller stretch being slightly faster, as the BFS depth depends on α . The maximum time for Steinlib is 14 seconds.

Comparison. Table 3 shows the percentage of instances an algorithm can solve strictly faster than another algorithm, as well as the average relative speed-up factor in these cases. An example is provided in Fig. 2 for unweighted instances and $\alpha = 5$. Clearly, BBMR is the slowest of the considered algorithms. Only KP was sometimes slower (in 8.2% of the cases, roughly 14-fold); but when KP was faster, it was so by a factor of over 116. BS shows a significant number of instances where it is fastest, as it benefits from traversing adjacency lists instead of calculating distances. Also, its speed-up factor in these cases is rather strong. EN and ADDJS both share a similar number of instances each can solve faster. This is no surprise since EN's BFS for each node is similar to the Dijkstra runs for ADDJS in the unweighted setting. Only for large or dense instances, ADDJS gets noticeably faster than EN (cf. Fig. 2). If these values were to be restricted to larger or denser graphs (e.g. $n \geq 100$ or $\varrho_r \geq 5\%$), the results would be even clearer due to the fact that all small/sparse instances can be solved within fractions of seconds by most algorithms.

■ **Table 3** Percentage of instances where A was strictly faster than B. “–” denotes that the algorithms cannot be directly compared due to their incompatible stretch restrictions (2 vs. odd), see Table 1. The speed-up factor is given in parentheses. Only instances both algorithms can solve are included.

A \ B	ADDJS	BS	KP	BBMRY	EN
ADDJS		14.83% (1.45)	51.23% (123.61)	93.24% (210.88)	19.38% (1.07)
BS	36.07% (24.71)		–	92.99% (188.60)	28.70% (30.23)
KP	1.19% (1.27)	–		86.65% (116.61)	–
BBMRY	0.00% (–)	0.00% (–)	8.20% (14.17)		0.00% (–)
EN	27.10% (2.68)	11.52% (0.45)	–	93.37% (170.12)	

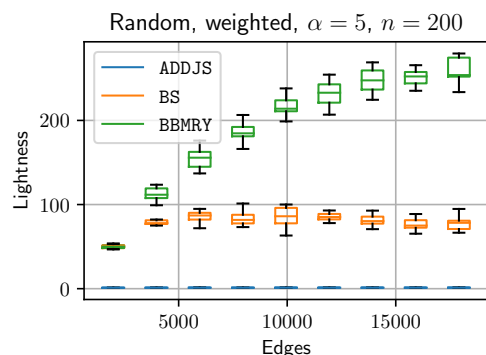
4.2 Quality

We may now consider the standard measures for spanners: their lightness and sparseness.

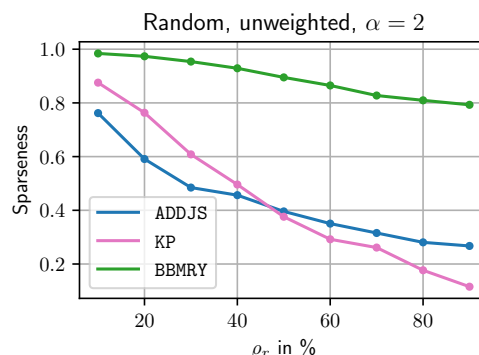
Weight functions for the Random instances. A key observation from the running times above also holds true for the solutions’ quality: The unweighted instances and the weighting **W1** behave very similarly. In any measure, **W1** lies between **W2** and the unweighted case, but is always very close to the latter; with increasing graph size, the sparseness of **W1** converges to that of the latter. Thus, in the following discussions regarding Random, we focus on unweighted instances and **W2**; all statements about the former are also true for **W1**.

Lightness. As a concept different from size, the lightness is only relevant for weighted instances. **BBMRY** and **BS** do not provide any guarantees, and indeed, both yield arbitrarily high lightness values in practice across all instance libraries (cf. Fig. 3). **BBMRY**’s lightness is on average 33.5 times higher than **ADDJS**’s; and **BS**’s lightness is 11.8 times higher. **BS** and **BBMRY** never yield a lower lightness than **ADDJS** on any instance. In summary, there is no alternative to **ADDJS** for applications requiring a lightness guarantee.

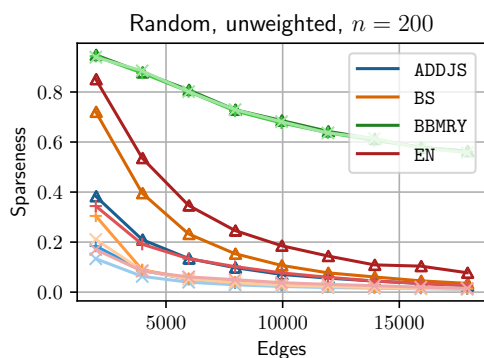
Sparseness. First, consider $\alpha = 2$ and unweighted graphs. Surprisingly, **KP** yields a lower sparseness than **ADDJS** for $q_r \gtrsim 50\%$ (cf. Fig. 4). **BBMRY** yields only high sparseness in comparison to those algorithms. On **Tsplib** (i.e., complete graphs), **KP** always finds the trivial optimal solution of a star due to the way the algorithm works. In contrast, **KP** can only sparsify 22% of the Steinlib instances in comparison to 78% by **ADDJS** and 62% by **BBMRY**.



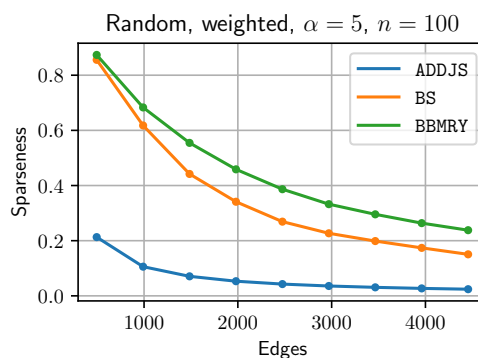
■ **Figure 3** Lightness for **ADDJS**, **BBMRY** and **BS**. **ADDJS** has a maximum lightness of 1.3.



■ **Figure 4** Sparseness comparison averaged on all instances from Random.



■ **Figure 5** Sparseness for unweighted graphs with different stretches: $\alpha = 3$ (Δ), $\alpha = 5$ (+), and $\alpha = 7$ (\times).



■ **Figure 6** Sparseness for weighted graphs. BBMRV and BS have significantly higher sparseness than ADDJS.

For unweighted instances with odd stretch, ADDJS, BS, and EN behave similarly, see Fig. 5. For dense graphs, the sparseness converges and higher stretches unsurprisingly result in lower sparseness. ADDJS generally yields the lowest sparseness, followed by BS and then EN. E.g., over all libraries for $\alpha = 5$, BS yields 37% higher sparseness than ADDJS, and EN 47% higher than BS. Although BBMRV’s sparseness seems stretch-independent, it is very high compared to all other algorithms. EN shows a higher variance of the sparseness; e.g. for $\alpha = 3$ its standard deviation is about 4 times that of ADDJS. With higher stretches, the variance decreases.

On weighted graphs, BBMRV and BS yield comparable sparseness; both are, however, significantly worse than ADDJS, see Fig. 6. All algorithms except for ADDJS cannot significantly sparsify the already sparse graphs ($\varrho_r \leq 10\%$) from Steinlib too much: ADDJS’s average sparseness is 71.7%, while the sparsenesses of all other algorithms are between 92% and 97%.

Effective stretch. For instances with $n \leq 1000$ ($n \leq 11000$ for Steinlib), we computed an APSP on the input graphs and the resulting spanners to calculate the effective stretches. More precisely, for each vertex pair (u, v) , we can compare the length of the shortest path in the input graph to that in the spanner, and obtain an effective stretch specific to (u, v) . We can aggregate over the effective stretches for all vertex pairs. Table 4 provides an overview of the effective mean, mean maximum, and maximum stretches over all instances. ADDJS gives the overall highest mean stretch and thus always utilizes the limit provided by α . Interestingly, BBMRV, BS, and EN never use the available stretch of 7 for unweighted graphs. For unweighted complete graphs, EN does not yield a mean stretch over 2 (cf. [15, Appendix, Figure 11]), regardless of α . KP behaves similar to ADDJS and even yields a higher mean stretch on those instances both ADDJS and KP could solve. BBMRV and BS yield lower mean stretches in comparison to ADDJS for weighted graphs. In summary, these results directly correlate to the previous observations. By having a low mean stretch and not utilizing the allowed stretch, it is clear that some potential for sparsification is not fully exploited, so the lightness and sparseness are also worse.

4.3 Further Properties for Weighted Spanners

Finally, we want to touch on the graph structure of spanners particularly on *weighted graphs*. We are interested in their mean degrees, and the number of edges in the shortest paths.

■ **Table 4** Effective mean (mean max, max) stretch for multiple configurations.

	α	ADDJS	BS	BBMR	KP/EN
unweighted	2	1.36 (1.98, 2.00)	–	1.10 (1.86, 2.00)	KP: 1.36 (1.87, 2.00)
	3	1.77 (2.94, 3.00)	1.53 (2.77, 3.00)	1.12 (2.06, 3.00)	EN: 1.43 (2.61, 3.00)
	4	2.16 (3.89, 4.00)	–	1.13 (2.13, 4.00)	–
	5	2.47 (4.77, 5.00)	1.77 (3.68, 5.00)	1.13 (2.11, 4.00)	EN: 1.59 (3.13, 5.00)
	7	2.86 (6.43, 7.00)	1.83 (4.16, 6.00)	1.13 (2.12, 5.00)	EN: 1.73 (3.52, 6.00)
weighted	2	1.03 (1.85, 2.00)	–	1.00 (1.19, 2.00)	–
	3	1.07 (2.70, 3.00)	1.01 (1.16, 2.61)	1.00 (1.65, 3.00)	–
	4	1.11 (3.49, 4.00)	–	1.01 (1.99, 4.00)	–
	5	1.16 (4.20, 5.00)	1.01 (1.37, 3.26)	1.01 (2.30, 5.00)	–
	7	1.25 (5.44, 7.00)	1.01 (1.60, 3.49)	1.01 (2.69, 7.00)	–

Degree. For $\alpha = 3$, BS yields a higher mean degree by 30% than BBMR, but is slightly lower for $\alpha \geq 5$; BS is the only algorithm where a higher stretch results in lower mean degrees. Overall, BBMR and BS yield very high mean degrees (similar to the lightness and sparseness) while ADDJS has mean degrees around 3–5. For the Steinlib instances, ADDJS’s spanners yields a mean degree of (on average) 57% of the mean degree of the input graph. In contrast to this, all other algorithms’ spanners have more than 96% of the original mean degree, see [15, Appendix, Figure 10]. All algorithms show a constant or moderately (linearly) increasing dependency between mean degrees of the original graph and the spanner, except for KP. There, the spanners’ mean degrees first increase together with the original mean degree, but for instances with $\rho_r \gtrsim 50\%$ the former decrease again. Analogous to the sparseness, the mean degrees become lower than ADDJS’s for large relative densities.

Hops. Lastly, we take a look at the number of edges (*hops*) of all shortest paths (again considering the instances described for the effective stretch). Overall, the mean hop difference is always positive, i.e., shortest paths in the spanner on average use more edges than in the original graph. Only 2.27% of all vertex pairs have shortest paths with fewer hops in the spanner than in the original graph; 58.90% retain their hop count, and 38.82% gain at least one hop. ADDJS has a higher average mean hop difference of 3.98 than BS (0.26) and BBMR (0.15). The tendency for all algorithms is to yield a higher hop difference with higher stretch.

5 Conclusions and Questions

We conducted the first experimental evaluation of polynomial spanner approximations. We can provide a rough guideline on which algorithm to use. If one has the special case of $\alpha = 2$ and no edge weights, KP may be used for graphs with $\rho_r \gtrsim 50$ as long as it can solve the instances in feasible time. In all other cases ADDJS (the oldest and most simplistic approach!) should be the algorithm of choice, as long as the graphs have reasonable sizes. It provides the sparsest and lightest spanners within a reasonable running time. Especially for weighted graphs, no other algorithm can calculate spanners of comparable quality. Only for very large unweighted graphs, BS is a good alternative to ADDJS due to its low running time. BBMR’s subpar performance makes its only sensible for directed graphs, as it is the only algorithm capable of handling such instances. Lastly, while EN can produce good spanners, it is never strictly better in running time nor quality than ADDJS or BS.

Overall, the practical strength of the algorithmic idea of ADDJS seems to be established. Based thereon, it now seems worthwhile to investigate the practical performance of proposed improvements. In particular, this includes the mentioned implementation-wise more intricate

but theoretically faster variants [42, 45] which lose the lightness guarantees. Furthermore, we may ask whether ADDJS can be improved for unweighted graphs by using a specific edge order. Formally, an arbitrary order suffices, but using a BFS or DFS order, or sequentially considering sets of independent edges, may further decrease the spanner size in practice. It may also be interesting to investigate the algorithmic reasons for the significant running time degradation of ADDJS (and BBMY) for $\alpha = 2$ on unweighted graphs, and to find theoretical results complementing the practical findings on the measures considered in Sect. 4.3.


References

- 1 Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen Kobourov, and Richard Spence. Graph spanners: A tutorial review. *Computer Science Review*, 37:100253, 2020. doi:10.1016/j.cosrev.2020.100253.
- 2 Reyan Ahmed, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen Kobourov, Faryad Darabi Sahneh, and Richard Spence. Approximation algorithms and an integer program for multi-level graph spanners. In *Proc. SEA 2019*, volume 11544 of *LNCS*, pages 541–562. Springer, 2019. doi:10.1007/978-3-030-34029-2.
- 3 Stephen Alstrup, Søren Dahlgaard, Arnold Filtser, Morten Stöckel, and Christian Wulff-Nilsen. Constructing light spanners deterministically in near-linear time. In *Proc. ESA 2019*, volume 144 of *LIPICs*, pages 4:1–4:15. LZI Dagstuhl, 2019. doi:10.4230/LIPICs.ESA.2019.4.
- 4 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete Computational Geometry*, 9:81–100, 1993. doi:10.1007/BF02189308.
- 5 Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, 1985. doi:10.1145/4221.4227.
- 6 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007. doi:10.1002/rsa.20130.
- 7 Benchmark and experimental data. Url. <https://tcs.uos.de/research/spanner>, 2022.
- 8 Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Approximation algorithms for spanner problems and directed Steiner forest. *Information and Computation*, 222:93–107, 2013. doi:10.1016/j.ic.2012.10.007.
- 9 Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David P. Woodruff. Transitive-closure spanners. In *Proc. SODA 2009*, pages 932–941. ACM SIAM, 2009. doi:10.1137/110826655.
- 10 Quirijn W. Bouts, Alex P. ten Brink, and Kevin Buchin. A framework for computing the greedy spanner. In *Proc. SoCG 2014*, pages 11–19. ACM, 2014. doi:10.1145/2582112.2582154.
- 11 Leizhen Cai. NP-completeness of minimum spanner problems. *Discrete Applied Mathematics*, 48(2):187–194, 1994. doi:10.1016/0166-218X(94)90073-6.
- 12 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*, 42(1):67–90, 1995. doi:10.1145/200836.200853.
- 13 Barun Chandra, Gautam Das, Giri Narasimhan, and José Soares. New sparseness results on graph spanners. In *Proc. SoCG 1992*, pages 192–201. ACM, 1992. doi:10.1145/142675.142717.
- 14 Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The open graph drawing framework (ogdf). In Roberto Tamassia, editor, *Handbook of graph drawing and visualization*, chapter 17. CRC press, 2014.
- 15 Markus Chimani and Finn Stutzenstein. Spanner approximations in practice. *arXiv*, 2022. arXiv:2107.02018.
- 16 Kenneth Lee Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. STOC 1987*, pages 56–65. ACM, 1987. doi:10.1145/28395.28402.

- 17 Lenore J. Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001. doi:10.1006/jagm.2000.1134.
- 18 Lenore J. Cowen and Christopher G. Wagner. Compact roundtrip routing in directed networks. *Journal of Algorithms*, 50(1):79–95, 2004. doi:10.1016/j.jalgor.2003.08.001.
- 19 IBM ILOG Cplex. V12. 1: User’s manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009. URL: <https://www.ibm.com/analytics/cplex-optimizer>.
- 20 Michael Dinitz and Robert Krauthgamer. Directed spanners via flow-based linear programs. *arXiv*, 2010. arXiv:1011.3701.
- 21 Michael Dinitz and Zeyu Zhang. Approximating low-stretch spanners. In *Proc. SODA 2016*, pages 821–840. ACM SIAM, 2016. doi:10.1137/1.9781611974331.ch59.
- 22 Michael Elkin. An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. *SIAM Journal on Computing*, 36(2):433–456, 2006. doi:10.1137/S0097539704441058.
- 23 Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. *ACM Transactions on Algorithms*, 15(1), 2018. doi:10.1145/3274651.
- 24 Michael Elkin, Ofer Neiman, and Shay Solomon. Light spanners. *SIAM Journal on Discrete Mathematics*, 29(3):1312–1321, 2015. doi:10.1137/140979538.
- 25 Michael Elkin and David Peleg. Approximating k-spanner problems for $k > 2$. *Theoretical Computer Science*, 337(1):249–277, 2005. doi:10.1016/j.tcs.2004.11.022.
- 26 Michael Elkin and Shay Solomon. Fast constructions of lightweight spanners for general graphs. *ACM Transactions on Algorithms*, 12(3), 2016. doi:10.1145/2836167.
- 27 Mohammad Farshi and Joachim Gudmundsson. Experimental study of geometric t-spanners. *ACM Journal on Experimental Algorithmics*, 14, 2010. doi:10.1145/1498698.1564499.
- 28 Giorgio Gallo, Michael D. Grigoriadis, and Robert E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989. doi:10.1137/0218003.
- 29 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.
- 30 Andrew Vladislav Goldberg. Finding a maximum density subgraph. Technical Report UCB/CSD-84-171, EECS Department, University of California, Berkeley, 1984.
- 31 Madhav Jha and Sofya Raskhodnikova. Testing and reconstruction of Lipschitz functions with applications to data privacy. *SIAM Journal on Computing*, 42(2):700–731, 2013. doi:10.1137/110840741.
- 32 J. Mark Keil. Approximating the complete Euclidean graph. In *Proc. SWAT 1988*, volume 318 of *LNCS*, pages 208–213. Springer, 1988.
- 33 Thorsten Koch, Alexander Martin, and Stefan Voß. SteinLib: An updated library on Steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2000. URL: <http://steinlib.zib.de/steinlib.php>.
- 34 Guy Kortsarz and David Peleg. Generating sparse 2-spanners. *Journal of Algorithms*, 17(2):222–236, 1994. doi:10.1006/jagm.1994.1032.
- 35 Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proc. SPAA 2015*, pages 192–201. ACM, 2015. doi:10.1145/2755573.2755574.
- 36 Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proc. SPAA 2013*, pages 196–203. ACM, 2013. doi:10.1145/2486159.2486180.
- 37 David Peleg. Proximity-preserving labeling schemes and their applications. In *Proc. WG 1999*, volume 1665 of *LNCS*, pages 30–41. Springer, 1999.
- 38 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.
- 39 David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. In *Proc. PODC 1987*, pages 77–85. ACM, 1987. doi:10.1145/41840.41847.

- 40 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, 1989. doi:10.1145/65950.65953.
- 41 Gerhard Reinelt. TspLib—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991. doi:10.1287/ijoc.3.4.376.
- 42 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61:389–401, 2011.
- 43 H. Shpungin and M. Segal. Near optimal multicriteria spanner constructions in wireless ad-hoc networks. In *IEEE INFOCOM 2009*, pages 163–171, 2009. doi:10.1109/INFOCOM.2009.5061918.
- 44 Mikkel Sigurd and Martin Zachariasen. Construction of minimum-weight spanners. In *Proc. ESA 2004*, volume 3221 of *LNCS*, pages 797–808. Springer, 2004.
- 45 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

Determinants from Homomorphisms

Radu Curticapean  

IT University of Copenhagen, Denmark

Basic Algorithms Research Copenhagen, Denmark

Abstract

We give a new combinatorial explanation for well-known relations between determinants and traces of matrix powers. Such relations can be used to obtain polynomial-time and poly-logarithmic space algorithms for the determinant. Our new explanation avoids linear-algebraic arguments and instead exploits a classical connection between subgraph and homomorphism counts.

2012 ACM Subject Classification Computing methodologies → Linear algebra algorithms; Mathematics of computing → Combinatorics; Mathematics of computing → Graph theory; Theory of computation → Parallel algorithms

Keywords and phrases determinant, homomorphisms, matrix trace, Newton identities

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.38

Related Version *arXiv version*: <https://arxiv.org/abs/2204.10718>

Funding BARC is supported by Villum Foundation grant 16582.

1 Introduction

The determinant of $n \times n$ matrices is, up to scaling, the unique function from $n \times n$ matrices to scalars that is linear and alternating in the rows and columns. It admits the well-known *Leibniz formula*

$$\det(A) = \sum_{\pi \in S_n} \operatorname{sgn}(\pi) \prod_{i=1}^n a_{i, \pi(i)}, \quad (1)$$

where S_n denotes the set of permutations of $\{1, \dots, n\}$ and $\operatorname{sgn} : S_n \rightarrow \{-1, 1\}$ denotes the permutation sign. Writing $\sigma(\pi)$ for the number of cycles in π , the permutation sign can be expressed as $\operatorname{sgn}(\pi) = (-1)^{n+\sigma(\pi)}$.

When presented with only the right-hand side of (1), unaware of the connection to the determinant, one would likely struggle to evaluate this sum of $n!$ terms efficiently. For comparison, it is $\#\text{P}$ -hard to compute the similarly defined *permanent* [8], which is obtained by omitting the sign factors from the expression.

Yet, determinants can be evaluated efficiently, e.g., via Gaussian elimination in $O(n^3)$ field operations, including divisions. Asymptotically optimal algorithms achieve $O(n^\omega)$ operations, where $\omega < 3$ is the exponent of matrix multiplication [2, Exercise 28.2-3]. Note that $\det(A)$ is defined over any ring containing the entries of A ; there are also algorithms computing determinants with a polynomial number of ring operations, i.e., excluding divisions [6, 1, 7].

Determinants from matrix powers

It is classically known in linear algebra that $\det(A)$ for $n \times n$ matrices A can be computed from the matrix traces $\operatorname{tr}(A^k)$ for $1 \leq k \leq n$. In the following, assume that A is defined over an algebraically closed field \mathbb{F} , such that A has eigenvalues $\lambda_1, \dots, \lambda_n \in \mathbb{F}$. The idea is to express $\det(A)$ and $\operatorname{tr}(A^k)$ for $1 \leq k \leq n$ as particular polynomials in the eigenvalues $\lambda_1, \dots, \lambda_n$ and then relate these polynomials.



© Radu Curticapean;

licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 38; pp. 38:1–38:7

Leibniz International Proceedings in Informatics



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

- The determinant can be expressed as $\det(A) = \lambda_1 \dots \lambda_n$; this is the n -th elementary symmetric polynomial in the eigenvalues. Generally, the k -th elementary symmetric polynomial $e_k(x_1, \dots, x_n)$ in n variables is the sum of monomials $\sum_S \prod_{i \in S} x_i$, where S ranges over all k -subsets $S \subseteq \{1, \dots, n\}$.
- The matrix trace satisfies $\operatorname{tr}(A) = \lambda_1 + \dots + \lambda_n$, and more generally, $\operatorname{tr}(A^k) = \lambda_1^k + \dots + \lambda_n^k$. This is the k -th power-sum polynomial in the eigenvalues. Generally, the k -th power sum polynomial is $p_k(x_1, \dots, x_n) = x_1^k + \dots + x_n^k$.

The Girard–Newton identities then allow us to relate the power-sum and elementary symmetric polynomials. They state that, for all $1 \leq k \leq n$,

$$k e_k(x_1, \dots, x_n) = \sum_{i=1}^k (-1)^{i-1} e_{k-i}(x_1, \dots, x_n) p_i(x_1, \dots, x_n).$$

For \mathbb{F} of characteristic 0, a recursive application of these identities allows us to compute $\det(A) = e_n(\lambda_1, \dots, \lambda_n)$ from the values $p_k(\lambda_1, \dots, \lambda_n) = \operatorname{tr}(A^k)$ for $1 \leq k \leq n$. Csanky’s algorithm [4, Chapter 31] implements this approach with an arithmetic circuit of bounded fan-in, $O(\log^2 n)$ depth, and polynomial size. In other words, it shows that determinants can be computed with $O(\log^2 n)$ operations on a polynomial number of parallel processors.

Our result

The main result of this paper is a novel and self-contained derivation of a known and algorithmically useful formula that expresses the determinant of $n \times n$ matrices A as a polynomial combination of traces of matrix powers:

$$\det(A) = (-1)^n \sum_{\lambda \vdash n} (-1)^{|\lambda|} \prod_{\ell=1}^n \frac{\operatorname{tr}(A^\ell)^{s_\ell(\lambda)}}{s_\ell(\lambda)! \cdot \ell^{s_\ell(\lambda)}}. \quad (2)$$

Some remarks on the notation are in order. The sum ranges over partitions λ of n , which are multi-sets of positive numbers summing to n . We write $\lambda \vdash n$ to indicate that λ is a partition of n and write $|\lambda|$ for its number of parts. For $\ell \in \mathbb{N}$, we write $s_\ell(\lambda) \in \mathbb{N}$ for the number of occurrences of ℓ in λ .

The previous subsection essentially gives a proof of (2) by appropriately expanding the recursive applications of the Girard–Newton identities. The new proof we present in this paper bypasses notions like eigenvalues, symmetric polynomials, and the Girard–Newton identities, and instead relies on ideas from the theory of graph homomorphism counts.

Our proof of (2) is contained in Section 2. We then sketch in Section 3 how this formula can be used to obtain polynomial-time and parallel algorithms for the determinant.

2 Proof of Equation (2)

In the following, let $[n] = \{1, \dots, n\}$ and let $A = (a_{i,j})_{i,j \in [n]}$ be a matrix. We will study the determinant of A using graph-theoretic language. The graphs G we consider are *directed* and may feature *self-loops*, and some graphs may feature *parallel edges* between the same pair of vertices. We write $V(G)$ and $E(G)$ for the vertices and edges of G .

2.1 Determinants are sums of cycle covers

The matrix A induces an edge-weighted complete directed graph with self-loops on the vertex set $V(A) = [n]$. Abusing notation, we also write A for this weighted graph. In this view, permutations correspond to *cycle covers*, which are edge-sets $C \subseteq E(A)$ inducing vertex-disjoint cycles that cover all vertices of A .

We require the more general notion of a k -*partial cycle cover* for $0 \leq k \leq n$, which is a collection of vertex-disjoint cycles with k edges in total. We write $\sigma(C)$ for the number of cycles in C , define the sign of C analogously to the permutation sign as $\text{sgn}(C) = (-1)^{|C|+\sigma(C)}$, and define the *format* of C as the partition $\lambda \vdash k$ induced by the multi-set of cycle lengths. Finally, let $\mathcal{C}(n, k)$ be the set of k -partial cycle covers of the complete directed graph on vertex set $[n]$.

Partial cycle covers are connected to k -*partial determinants*. Up to sign, these are the coefficients of characteristic polynomials, and they can be defined as

$$\det_k(A) = \sum_{S \subseteq [n] \text{ of size } k} \det(A[S]),$$

where $A[S]$ is the square sub-matrix of A defined by restricting to the rows and columns contained in S . From the Leibniz formula (1), it follows that

$$\det_k(A) = \sum_{C \in \mathcal{C}(n, k)} \text{sgn}(C) \prod_{uv \in C} a_{u,v}. \quad (3)$$

Given $\lambda \vdash k$, let $C_\lambda \in \mathcal{C}(n, k)$ be any fixed cycle cover of format λ . We can regroup terms in (3) to obtain

$$\det_k(A) = \sum_{\lambda \vdash k} \text{sgn}(C_\lambda) \underbrace{\sum_{\substack{C \in \mathcal{C}(n, k) \text{ of} \\ \text{format } \lambda}} \prod_{uv \in C} a_{u,v}}_{=: \text{sub}(C_\lambda \rightarrow A)}. \quad (4)$$

Note that the quantity $\text{sub}(C_\lambda \rightarrow A)$ defined above is a weighted sum over the cycle covers C isomorphic to C_λ , weighted by the product of the edge-weights in C .

2.2 Relating subgraphs, embeddings and homomorphisms

Let L be a graph, possibly containing parallel edges. The weighted *homomorphism* and *embedding* counts from L into A are defined as

$$\text{hom}(L \rightarrow A) = \sum_{f: V(L) \rightarrow [n]} \prod_{\substack{e \in E(L) \\ \text{with } e=uv}} a_{f(u), f(v)}, \quad (5)$$

$$\text{emb}(L \rightarrow A) = \sum_{\substack{f: V(L) \rightarrow [n] \\ \text{injective}}} \prod_{\substack{e \in E(L) \\ \text{with } e=uv}} a_{f(u), f(v)}. \quad (6)$$

For example, if C_ℓ denotes the directed ℓ -cycle for $\ell \in \mathbb{N}$, then $\text{hom}(C_\ell \rightarrow A) = \text{tr}(A^\ell)$. Moreover, for $\lambda \vdash k$, recall that $s_\ell(\lambda)$ counts the occurrences of part ℓ in λ . We have

$$\text{hom}(C_\lambda \rightarrow A) = \prod_{\ell=1}^k \text{tr}(A^\ell)^{s_\ell(\lambda)}, \quad (7)$$

since homomorphisms from a disjoint union of graphs can be chosen independently for the individual components; this implies that $\text{hom}(C_\lambda \rightarrow A)$ is the product of homomorphism counts for the individual cycles in C_λ .

Given a graph P without parallel edges, an *automorphism* of P is an isomorphism into itself. We write $\text{aut}(P)$ for the number of automorphisms of P . For example,

$$\text{aut}(C_\lambda) = \prod_{\ell=1}^k s_\ell(\lambda)! \cdot \ell^{s_\ell(\lambda)}, \quad (8)$$

38:4 Determinants from Homomorphisms

since any automorphism of C_λ (i) permutes the set of $s_\ell(\lambda)$ cycles for any fixed length ℓ , which gives rise to a factor of $s_\ell(\lambda)!$ in the above product, and (ii) independently applies an automorphism to each cycle, giving rise to a factor of ℓ for every cycle of length ℓ .

With these notions set up, we can successively express subgraph counts from cycle covers, as defined in (4), via homomorphism counts. First, we transition from subgraph to embedding counts: As every subgraph isomorphic to C_λ gives rise to $\text{aut}(C_\lambda)$ many embeddings with the same image, we obtain

$$\text{sub}(C_\lambda \rightarrow A) = \frac{\text{emb}(C_\lambda \rightarrow A)}{\text{aut}(C_\lambda)}. \quad (9)$$

Next, we transition from embedding to homomorphism counts. Roughly speaking, embedding counts from a graph H are equal to homomorphism counts from H plus “lower-order terms” involving only homomorphism counts from graphs F with strictly less vertices than H . This follows directly from [5, (5.18)] and we include a simple proof for completeness, also contained in [3].

► **Lemma 1.** *For any fixed graph H , there are coefficients $\beta_F \in \mathbb{Z}$ for all graphs F with $|V(F)| < |V(H)|$ such that*

$$\text{emb}(H \rightarrow A) = \text{hom}(H \rightarrow A) + \sum_{\substack{\text{graphs } F \text{ with} \\ |V(F)| < |V(H)|}} \beta_F \text{hom}(F \rightarrow A).$$

Proof. Given a partition ρ of the set¹ $V(H)$, the *quotient* H/ρ is the multigraph obtained by identifying the vertices within each block of ρ while keeping all possibly emerging self-loops and multi-edges. We have

$$\text{hom}(H \rightarrow A) = \sum_{\substack{\text{partition } \rho \\ \text{of } V(H)}} \text{emb}(H/\rho \rightarrow A), \quad (10)$$

since any homomorphism $f : V(H) \rightarrow [n]$ induces a partition $\rho = \{f^{-1}(i) \mid i \in [n]\}$ by which f may be viewed as an embedding from H/ρ to A .

Write \perp for the finest partition of the set $V(H)$, that is, the partition consisting of $|V(H)|$ singleton parts. By rearranging (10) and using $H/\perp = H$, we obtain

$$\text{emb}(H \rightarrow A) = \text{hom}(H \rightarrow A) - \sum_{\substack{\text{partition } \rho \neq \perp \\ \text{of } V(H)}} \text{emb}(H/\rho \rightarrow A). \quad (11)$$

Note that all graphs H/ρ with $\rho \neq \perp$ have strictly less vertices than H . We can therefore apply (11) again to express each term $\text{emb}(H/\rho \rightarrow A)$ on the right-hand side as $\text{hom}(H/\rho \rightarrow A)$ minus embedding counts from smaller graphs. This process can be iterated until reaching single-vertex graphs, from which homomorphism and embedding counts coincide trivially. Upon termination of this process, all occurrences of embedding counts have been replaced by homomorphism counts. ◀

Combining (4), (9), and Lemma 1, it follows that the k -partial determinant is a linear combination of homomorphism counts from k -partial cycle covers plus “lower-order terms”.

¹ Note that ρ here is a partition of a *set*, not a partition of a *number*.

► **Corollary 2.** For any fixed $k \in \mathbb{N}$, there are coefficients $\alpha_F \in \mathbb{Q}$ for all graphs F with $|V(F)| < k$ such that, for any $n \times n$ matrix A ,

$$\det_k(A) = \left(\sum_{\lambda \vdash k} \frac{\text{sgn}(C_\lambda)}{\text{aut}(C_\lambda)} \text{hom}(C_\lambda \rightarrow A) \right) + \sum_{\substack{F \text{ with} \\ |V(F)| < k}} \alpha_F \text{hom}(F \rightarrow A). \quad (12)$$

2.3 Lower-order terms vanish

As it turns out, the “lower-order terms” in (12) vanish. To show this, we use Kronecker products to lift this equality to a polynomial identity and then compare coefficients. For $t \in \mathbb{N}$, the *Kronecker product*² $A \otimes J_t$ of A with the $t \times t$ all-ones matrix J_t is an $nt \times nt$ matrix with row and column indices from $[n] \times [t]$, such that the entry at row (i, r) and column (j, r') equals $a_{i,j}$. In other words, each entry $a_{i,j}$ of A is replaced in $A \otimes J_t$ by a $t \times t$ matrix that contains only $a_{i,j}$.

It turns out that (12) “reacts polynomially” to this operation: When fixing $k = n$ and replacing A by $A \otimes J_t$ for varying t in (12), the homomorphism counts $\text{hom}(S \rightarrow A \otimes J_t)$ for graphs S on the right-hand side become polynomials in t . In fact, each such homomorphism count contains only a single monomial:

► **Claim 3.** For any graph S , we have $\text{hom}(S \rightarrow A \otimes J_t) = t^{|V(S)|} \text{hom}(S \rightarrow A)$.

Proof. Every function $f : V(S) \rightarrow [n]$ induces $t^{|V(S)|}$ functions $f' : V(S) \rightarrow [n] \times [t]$, all of the same edge-weight product, by choosing an index $r_v \in [t]$ for each vertex $v \in V(S)$. Conversely, every such function is induced by the function that forgets the second component of the images. ◁

Applying Claim 3 on (12) with $k = n$, and with $A \otimes J_t$ instead of A , we obtain

$$\det_n(A \otimes J_t) = t^n \left(\sum_{\lambda \vdash n} \frac{\text{sgn}(C_\lambda)}{\text{aut}(C_\lambda)} \text{hom}(C_\lambda \rightarrow A) \right) + \sum_{\substack{F \text{ with} \\ |V(F)| < n}} t^{|V(F)|} \alpha_F \text{hom}(F \rightarrow A). \quad (13)$$

We now observe that $\det_n(A \otimes J_t)$ is proportional to t^n . This will allow us to ignore the lower-order graphs F in (13).

► **Claim 4.** We have $\det_n(A \otimes J_t) = t^n \det(A)$.

Proof. By definition of the partial determinant, we have

$$\det_n(A \otimes J_t) = \sum_{\substack{S \subseteq [n] \times [t] \\ \text{with } |S|=n}} \det((A \otimes J_t)[S]).$$

If S contains two pairs that agree in the first component, then the $n \times n$ matrix $(A \otimes J_t)[S]$ contains two equal columns and its determinant vanishes. We can therefore restrict the summation to index sets of the form $S = \{(1, r_1), \dots, (n, r_n)\}$ for $r_1, \dots, r_n \in [t]$. There are t^n sets S of this form, each with $(A \otimes J_t)[S] = A$. The claim follows. ◁

² The Kronecker product $A \otimes B$ can be defined for general A and B , but we only require $B = J_t$.

38:6 Determinants from Homomorphisms

Now consider the polynomial identity (13) again. By Claim 4, the left-hand side equals $t^n \det(A)$. Then comparing the coefficients of t^n on both sides yields

$$\det(A) = \sum_{\lambda \vdash n} \frac{\text{sgn}(C_\lambda)}{\text{aut}(C_\lambda)} \text{hom}(C_\lambda \rightarrow A) \quad (14)$$

$$= \sum_{\lambda \vdash n} (-1)^{n+|\lambda|} \prod_{\ell=1}^n \frac{\text{tr}(A^\ell)^{s_\ell(\lambda)}}{s_\ell(\lambda)! \cdot \ell^{s_\ell(\lambda)}}. \quad (15)$$

For the last equation, we expanded $\text{hom}(C_\lambda \rightarrow A)$ via (7) and $\text{aut}(C_\lambda)$ via (8), and we used the definition of $\text{sgn}(C_\lambda)$. This proves (2).

► **Remark.** The above argument applies more generally. Consider any function $F : \mathbb{Q}^{k \times k} \rightarrow \mathbb{Q}$ that admits a set \mathcal{H} of k -vertex graphs and coefficients $\alpha_H \in \mathbb{Q}$ for $H \in \mathcal{H}$ such that

$$F(A) = \sum_{H \in \mathcal{H}} \alpha_H \text{emb}(H \rightarrow A).$$

By Lemma 1, every $\text{emb}(H \rightarrow A)$ is a sum of $\text{hom}(H \rightarrow A)$ and lower-order homomorphism counts; this yields an analogue of (12). If F vanishes on $k \times k$ matrices with two identical rows/columns, then an analogue of Claim 4 holds, and it follows as above that the lower-order homomorphism counts vanish.

3 Algorithmic applications

Equation (2) does not directly imply a polynomial-time algorithm for the determinant, as the sum over partitions $\lambda \vdash n$ involves a super-polynomial number of terms. Nevertheless, this sum can be computed in polynomial time via dynamic programming or polynomial multiplication, as shown below.

► **Lemma 5.** *Given $\text{tr}(A^\ell)$ for all $1 \leq \ell \leq n$, we can compute $\det(A)$ with $O(n^3)$ operations.*

Proof. Let X be a formal indeterminate. For $1 \leq \ell \leq n$, define the polynomial

$$p_\ell(X) = \sum_{i=0}^{\lfloor n/\ell \rfloor} (-1)^i \frac{\text{tr}(A^\ell)^i}{i! \cdot \ell^i} X^{\ell i}.$$

We observe first that the coefficient of X^n in the product $(-1)^n p_1 \dots p_n$ is the desired sum in (2), and then focus on computing that coefficient.

For the first part, note that the coefficient of X^n can be viewed as a weighted count of all ways to choose a power X^{i_ℓ} from each polynomial p_ℓ , subject to $\sum_\ell \ell \cdot i_\ell = n$. These choices yield a partition $(1^{i_1}, \dots, n^{i_n}) \vdash n$ that is weighted by $\prod_{\ell=1}^n (-1)^{i_\ell} \frac{\text{tr}(A^\ell)^{i_\ell}}{i_\ell! \cdot \ell^{i_\ell}}$. Thus, the coefficient of X^n in $(-1)^n p_1 \dots p_n$ can be viewed as a sum over partitions $\lambda \vdash n$ whose terms correspond to those in (2).

We compute the first $n+1$ coefficients of $(-1)^n p_1 \dots p_n$, including the coefficient of X^n , by iteratively multiplying p_t onto $p_1 \dots p_{t-1}$ and truncating the intermediate result to the first $n+1$ coefficients. Using standard polynomial multiplication, each of the n iterations takes $O(n^2)$ operations. Overall, this procedure requires $O(n^3)$ operations. ◀

By naively iterating matrix multiplication, we can compute $\text{tr}(A^\ell)$ for all $1 \leq \ell \leq n$ with $O(n^{\omega+1})$ overall operations, where ω is the exponent of matrix multiplication. This implies:

► **Theorem 6.** *The determinant $\det(A)$ can be computed with $O(n^{\omega+1})$ operations.*

The traces and subsequent application of (2) can also be computed with arithmetic circuits of constant fan-in and poly-logarithmic depth: Any product of two matrices can be computed trivially in $O(\log n)$ depth and $O(n^3)$ size. Repeated squaring allows us to compute all matrix powers A^ℓ and their traces $\text{tr}(A^\ell)$ for $1 \leq \ell \leq n$ in $O(\log^2 n)$ depth and $\tilde{O}(n^4)$ overall size.

After all traces are computed, the polynomial multiplications from the proof of Lemma 5 can be performed in $O(\log^2 n)$ depth and $\tilde{O}(n^3)$ size: A single polynomial multiplication can be computed in $O(\log n)$ depth and $\tilde{O}(n^2)$ size. The truncation of the product $(-1)^n p_1 \dots p_n$ to the first $n + 1$ coefficients can then be computed as an $O(\log n)$ -depth binary tree, with p_1, \dots, p_n at the leaves, and each internal node performing a polynomial multiplication followed by truncating to the first $n + 1$ coefficients. This implies:

► **Theorem 7.** *The determinant $\det(A)$ can be computed with an arithmetic circuit of constant fan-in, $O(\log^2 n)$ depth, and $\tilde{O}(n^4)$ size.*

References

- 1 Stuart J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inf. Process. Lett.*, 18(3):147–150, 1984. doi:10.1016/0020-0190(84)90018-8.
- 2 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>.
- 3 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.*, pages 210–223. ACM, 2017. doi:10.1145/3055399.3055502.
- 4 Dexter C. Kozen. *Csanky's Algorithm*, pages 166–170. Springer New York, New York, NY, 1992. doi:10.1007/978-1-4612-4400-4_31.
- 5 László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, 2012. URL: <http://www.ams.org/bookstore-getitem/item=COLL-60>.
- 6 Meena Mahajan and V. Vinay. A combinatorial algorithm for the determinant. In Michael E. Saks, editor, *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 5-7 January 1997, New Orleans, Louisiana, USA*, pages 730–738. ACM/SIAM, 1997. URL: <http://dl.acm.org/citation.cfm?id=314161.314429>.
- 7 P. A. Samuelson. A method of determining explicitly the coefficients of the characteristic equation. *The Annals of Mathematical Statistics*, 13(4):424–429, 1942. URL: <http://www.jstor.org/stable/2235845>.
- 8 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

Conditional Lower Bounds for Dynamic Geometric Measure Problems

Justin Dallant  

Université libre de Bruxelles, Belgium

John Iacono  

Université libre de Bruxelles, Belgium

Abstract

We give new polynomial lower bounds for a number of dynamic measure problems in computational geometry. These lower bounds hold in the Word-RAM model, conditioned on the hardness of either 3SUM, APSP, or the Online Matrix-Vector Multiplication problem [Henzinger et al., STOC 2015]. In particular we get lower bounds in the incremental and fully-dynamic settings for counting maximal or extremal points in \mathbb{R}^3 , different variants of Klee’s Measure Problem, problems related to finding the largest empty disk in a set of points, and querying the size of the i ’th convex layer in a planar set of points. We also answer a question of Chan et al. [SODA 2022] by giving a conditional lower bound for dynamic approximate square set cover. While many conditional lower bounds for dynamic data structures have been proven since the seminal work of Pătraşcu [STOC 2010], few of them relate to computational geometry problems. This is the first paper focusing on this topic. Most problems we consider can be solved in $O(n \log n)$ time in the static case and their dynamic versions have only been approached from the perspective of improving known upper bounds. One exception to this is Klee’s measure problem in \mathbb{R}^2 , for which Chan [CGTA 2010] gave an unconditional $\Omega(\sqrt{n})$ lower bound on the worst-case update time. By a similar approach, we show that such a lower bound also holds for an important special case of Klee’s measure problem in \mathbb{R}^3 known as the Hypervolume Indicator problem, even for amortized runtime in the incremental setting.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases Computational geometry, Fine-grained complexity, Dynamic data structures

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.39

Related Version *Full Version*: <https://arxiv.org/abs/2112.10095> [28]

Funding *Justin Dallant*: 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.

Acknowledgements We thank Jean Cardinal for helpful discussions about the topic of this paper.

1 Introduction

In 1995, Gajentaan and Overmars [30] introduced the notion of 3SUM hardness, showing that a number of problems in computational geometry can not be solved in subquadratic time, assuming the so-called 3SUM problem can not be solved in subquadratic time.¹ The general approach of proving polynomial lower bounds based on a few conjectures about key problems has since grown into its own subfield of complexity theory known as *fine-grained*

¹ In 2014, Grønlund and Pettie [31] showed that the 3SUM problem can be solved in (slightly) subquadratic time. The modern formulation thus replaces “subquadratic” with “truly subquadratic”, i.e. $O(n^{2-\varepsilon})$ for some constant $\varepsilon > 0$.



complexity. The most popular of these conjectures concern the aforementioned 3SUM problem, All-Pairs-Shortest-Paths (APSP), Boolean Matrix Multiplication (BMM), Triangle finding in a graph, Boolean Satisfiability (SAT) and the Orthogonal Vectors problem (2OV) (see for example the introductory surveys by Bringmann [13] and V. V. Williams [55]). Another problem which crops up as a bottleneck in computational geometry is Hopcroft’s problem (see the recent paper by Chan and Zheng [24]).

Pătraşcu [50] launched the study of such polynomial lower bounds for dynamic problems, where instead of simply computing a function on a single input, we want to be able to update that input and get the corresponding output of the function without having to recompute it from scratch. In particular, he introduced the Multiphase problem and showed a polynomial lower bound on its complexity, conditioned on the hardness of the 3SUM problem. Using the Multiphase problem as a stepping stone, he showed conditional hardness results for a variety of dynamic problems. Improvements and other conditional lower bounds for dynamic problems (data structure problems) have since appeared in the literature [1–3, 5–7, 9, 11, 12, 25, 27, 34, 35, 38, 40, 41, 49, 53]. Of particular interest for the purpose of this work is a paper by Kopelowitz et al. [41] where the approach of Pătraşcu is improved by showing a tighter reduction from 3SUM to the so-called Set Disjointness problem (an intermediate problem between 3SUM and the Multiphase problem), as well as a paper by V. V. Williams and Xu [56], which obtains a similar reduction from the so-called Exact Triangle problem. Also particularly relevant here is the work of Henzinger et al. [34], who show that many of the known bounds on dynamic problems can be derived (and even strengthened) by basing proofs on a hardness conjecture about the Online Boolean Matrix-Vector Multiplication (OMv) problem which they introduce.

While computational geometry was one of first fields where conditional lower bounds for algorithms were applied, for example by showing that determining if a point set is in general position is 3SUM hard [30], the progress in conditional lower bounds for dynamic problems has not found widespread application to computational geometry; recent work has been largely confined to improved upper bounds. The only examples before the first version of this paper² relate to (approximate) nearest-neighbor search under different metrics (see the paper by Rubinfeld [51], the introductory article by Bringmann [14] as well as a preprint by Ko and Song [39]), a paper by Lau and Ritossa [44] with results for orthogonal range update on weighted point sets and an (unconditional) lower bound by Chan [17] for a dynamic version of Klee’s Measure Problem. After a previous version of the present paper appeared on arXiv, and independent of our work, Jin and Xu [37] studied generalized versions of the OMv and BMM problems and proved polynomial lower bounds for various dynamic problems based on their hardness, among which Dynamic 2D Orthogonal Range Color Counting, Counting Maximal Points, Dynamic Klee’s measure problem for unit hypercubes and Chan’s Halfspace Problem.

In this work, we exploit the results of Pătraşcu, Kopelowitz et al., V. V. Williams and Xu, and Henzinger et al. to give conditional polynomial lower bounds for a variety of dynamic problems in computational geometry, based on the hardness of 3SUM, APSP and Online Boolean Matrix-Vector Multiplication. Almost all the problems we study here share the common characteristic of being about computing a single global metric for a set of objects in space subject to updates. Moreover, in the static case (where there are no updates) most of these metrics can be computed in worst-case $O(n \log n)$ time using standard computational

² We exclude from this list examples where (conditional) bounds on the static case trivially imply polynomial bounds on the dynamic case.

geometry results. In particular, we show conditional hardness results for orthogonal range marking, maintaining the number of maximal or extremal points in a set of points in \mathbb{R}^3 , dynamic approximate square set cover, problems related to Klee’s Measure Problem, problems related to finding the largest empty disk in a set of points, testing whether a set of disks covers a given rectangle, and querying for the size of the i ’th convex layer of a set of points in the plane. We also give an unconditional lower bound for the incremental Hypervolume Indicator problem in \mathbb{R}^3 , where the goal is to maintain the volume of the union of a set of axis-aligned boxes which all have the origin as one of their vertices.

The most basic of these problems, and the one we present first, is Square Range Marking: given a set of n initially unmarked points in the plane, preprocess them to allow marking of the points in any given axis-aligned square and testing if there is any unmarked point. This encompasses the idea of augmenting a range query structure where augmentations can be applied to all data in a query range; a mark is the simplest such augmentation. While many variants of augmented orthogonal range queries have been studied (especially in the static case) [4, 21, 22, 32, 33, 36, 42, 45, 46, 52], this natural variant has been given little attention. This is perhaps no coincidence, as we show that the straightforward polynomial-time solution based on kd-trees is likely almost optimal, in contrast to standard 1-D range marking and other augmentation problems which are easily handled by suitable variants of BSTs [26, Ch. 14].

Lau and Ritossa [44] give similar lower bounds for data structures on weighted points, conditioned on the hardness of Online Boolean Matrix-Vector Multiplication, but explicitly leave open questions on points which have a color or a “category.” They show for example a lower bound for a data structure which allows to increment the weight of all points in an orthogonal range and to query the sum of weights for all points in a given range, as well as for variants of this problem.

1.1 Setting and computational model

We work in the standard Word RAM model, with words of $w = \Theta(\log n)$ bits unless otherwise stated, and for randomized algorithms we assume access to a perfect source of randomness. We will base our conditional lower bounds on the following well known hardness conjectures.

► **Conjecture 1** (3SUM conjecture). *The following problem (3SUM) requires $n^{2-o(1)}$ expected time to solve: given a set of n integers in $\{-n^3, \dots, n^3\}$, decide if three of them sum up to 0 .³*

► **Conjecture 2** (APSP conjecture). *The following problem (APSP) requires $n^{3-o(1)}$ expected time to solve: given an integer-weighted directed graph G on n vertices with no negative cycles, compute the distance between every pair of vertices in G .*

The 3SUM problem can easily be solved in $O(n^2)$ time, while APSP can be solved in cubic time by the Floyd–Warshall algorithm, for example. The best known methods improve these runtimes by subpolynomial factors [19, 54].

In addition to being the basis for these standard conjectures in fine-grained complexity, the 3SUM problem and the APSP problem are related in other ways (see [56]). In particular, they both fine-grained reduce to the Exact Triangle problem, meaning that if either the 3SUM conjecture or the APSP conjecture is true, then the following conjecture is true.

³ The assumption that the integers are in $\{-n^3, \dots, n^3\}$ is done without loss of generality. In the model we consider one can always reduce the problem to this setting while preserving the expected run-time, via known hashing methods [8].

► **Conjecture 3** (Exact Triangle conjecture). *The following problem (Exact Triangle) requires $n^{3-o(1)}$ expected time to solve: given an integer-weighted graph G and a target weight T , determine if there is a triangle in G whose edge weights sum to T .*

Thus, any bound conditioned on this conjecture also holds conditioned on the 3SUM conjecture or the APSP conjecture. We also consider a conjecture introduced by Henzinger et al. [34], which can be thought of as a weakening of the informal conjecture which says that “combinatorial” matrix multiplication on $n \times n$ matrices requires essentially cubic time (note that the term “combinatorial” is not well defined).

► **Conjecture 4** (OMv conjecture). *The following problem (OMv) requires $n^{3-o(1)}$ expected time to solve:*

We are given a $n \times n$ boolean matrix M . We can preprocess this matrix, after which we are given a sequence of n boolean column-vectors of size n denoted by v_1, \dots, v_n , one by one. After seeing each vector v_i , we must output the product Mv_i before seeing v_{i+1} .

The OMv problem can be solved in total time $O(n^3)$ by the naive algorithm. Here the best known method improves this runtime by a subpolynomial factor [43]. The conjecture was originally introduced in the Monte-Carlo setting (i.e. algorithms with a deterministic runtime but which are allowed to err with a small enough probability). We state it in the Las Vegas setting for the sake of uniformity of presentation. All the results of Henzinger et al. carry over to that setting with no difficulty.

While Henzinger et al. showed that most known lower bounds on dynamic problems derived from the 3SUM conjecture can be derived from the OMv conjecture (and often even strengthened), it is not known whether one conjecture implies the other. For most of our problems we derive polynomial lower bounds from both the OMv conjecture and the Exact Triangle conjecture. In such cases, we still get such lower bounds if at least one of the four considered conjectures is true. Moreover, the reductions used here could also give bounds in the case some of these conjecture fail by a small enough polynomial factor (for example if 3SUM requires $\Omega(n^{4/3})$ time).

Note also that recent work by Chan et al. [23] directly implies that the lower bounds we obtain from the APSP conjecture also hold in the so-called Real RAM model (conditioned on the analogous Real-APSP conjecture) and in restricted versions of the model. For the real versions of the 3SUM and Exact Triangle conjectures, combining our reductions with theirs would also imply polynomial lower bounds for many of the problems considered here, although weaker than the ones we obtain in the Word RAM model.

1.2 Main results

In the full version of this paper we obtain (conditional) polynomial lower bounds for a variety of dynamic geometric problems, and an unconditional bound for the incremental Hypervolume Indicator problem in \mathbb{R}^3 . Our bounds are stated as inequalities which imply trade-offs between achievable update and query times. The lower bounds we get on the maximum of both are summarized in Table 1, together with known upper bounds. Note that the bounds we get for squares or square ranges imply the same bounds for rectangles or general orthogonal ranges, although we sometimes get better trade-offs in these cases. Here we focus on some results for Square Range Marking, Counting Extremal Points in \mathbb{R}^3 and unweighted Square Set Cover, in the fully-dynamic setting. The other results can be found in the full version [28].

■ **Table 1** Non-trivial known upper bounds and new (at the time of the first version of this paper being made public) lower bounds on the maximum over update and query time derived from the Exact Triangle conjecture, the OMv conjecture or (in the case of the Hypervolume Indicator problem) unconditionally. The \tilde{O} notation hides polylog factors, while the O^* notation hides factors which are $o(n^\varepsilon)$ for an arbitrarily small constant $\varepsilon > 0$. All upper bounds are for data structures with at most $O^*(n)$ preprocessing. Note that the lower bounds for Square Range Marking also hold in the case of a static set of points (with some assumptions on preprocessing time) and that the lower bound for the Depth Problem derived from the OMv conjecture also holds for amortized runtime in the incremental setting. The lower bound obtained for counting maximal points has since been superseded by the more general result of Jin and Xu [37] who obtain lower bounds also in higher dimension.

Problem	Upper Bound	Lower Bound
Square Range Marking [§2.2,28]	$\tilde{O}(n^{1/2})^{\dagger,\ddagger}$ [15]	From Exact Triangle: $n^{1/4-o(1)^{\dagger}}$ From OMv: $n^{1/2-o(1)^{\dagger}}$
Counting Extremal Points in \mathbb{R}^3 [§3,28]	$O^*(n^{7/8})^{\dagger}$ [16]	From Exact Triangle: $n^{1/5-o(1)^{\dagger,\ddagger}}$ $n^{1/4-o(1)^{\ddagger,\S}}$ From OMv: $n^{1/2-o(1)^{\dagger,\ddagger}}$
Largest Empty Disk in Query Region [28]	$O^*(n^{11/12})^{\ddagger}$ [18]	
Largest Empty Disk in a Set of Disks [28]		
Rectangle Covering with Disks [28]		From OMv: $n^{1/2-o(1)^{\dagger,\ddagger}}$
Square Covering with Squares [28]	$\tilde{O}(n^{1/2})^{\ddagger}$ [57]	
Convex Layer Size in \mathbb{R}^2 [28]		
Counting Maximal Points in \mathbb{R}^3 [§3,28]	$\tilde{O}(n^{2/3})^{\ddagger}$ [18]	From Exact Triangle: $n^{1/4-o(1)^{\dagger}}$ $n^{1/3-o(1)^{\ddagger}}$
$O(n^\alpha)$ -approx. Weighted Square Set Cover [28]		From OMv: $n^{1/2-o(1)^{\dagger,\ddagger}}$
Klee's Measure Problem with Squares [28]	$\tilde{O}(n^{1/2})^{\ddagger}$ [57]	
Discrete KMP with Squares [28]	$O(n^{1/2})^{\dagger,\ddagger}$ [58]	
Depth Problem with Squares [28]	$\tilde{O}(n^{1/2})^{\ddagger}$ [57]	From Exact Triangle: $n^{1/3-o(1)^{\dagger,\ddagger}}$ From OMv: $n^{1/2-o(1)^{\dagger,\ddagger}}$
$O(1)$ -approximate Square Set Cover [§4,28]	$O^*(n^{1/2})^{\ddagger}$ [20]	From OMv: $n^{1/3-o(1)^{\dagger,\ddagger}}$
Hypervolume Indicator in \mathbb{R}^3 [28]	$\tilde{O}(n^{2/3})^{\ddagger}$ [18]	$\Omega(\sqrt{n})^{\#}$

[†] per-operation runtime in the incremental setting.

[‡] amortized runtime in the fully-dynamic setting.

[§] assuming $n^{1+o(1)}$ expected preprocessing time.

[#] unconditional lower bound in the incremental setting on amortized time, assuming at most polynomial time preprocessing, or on worst-case time without preprocessing assumptions.

Some of the lower bounds reveal interesting separations between geometric dynamic problems whose operations can be supported in subpolynomial or $O(n^\varepsilon)$ time and similar problems which require polynomial time with a fixed exponent (under the hardness conjectures we consider).

- Orthogonal range queries with dynamic updates on single points can be done with polylog time operations, while dynamic updates on orthogonal ranges of points require polynomial time.
- Dynamically maintaining maximal points in a point set can be done in polylog time in \mathbb{R}^2 , while maintaining only their number in \mathbb{R}^3 already requires polynomial time.
- The same separation between dimensions 2 and 3 applies for maintaining (the number of) extremal points.
- Related to the previous point, the ability to query for the size of any convex layer on a dynamic set of points in \mathbb{R}^2 requires polynomial time (compared to polylog time when we are only interested in the first convex layer, i.e. the convex hull).
- Maintaining a $O(1)$ -approximation for the size of dynamic unit square set cover can be done in $2^{O(\sqrt{\log n})}$ amortized time per update [20], while maintaining the size of a $O(n^\alpha)$ -approximation (for a constant $0 \leq \alpha < 1$) requires polynomial time for arbitrarily sized squares (with an exponent dependent on α).
- In the weighted case of the previous problem, we also get such a separation: $O(1)$ -approximate weighted unit square set cover can be done in $O(n^\varepsilon)$ time [20] while $O(n^\alpha)$ -approximate weighted dynamic square set cover requires polynomial time, with an exponent independent of α .

2 The general approach

In all the problems we consider, we have a data structure D which maintains a set S of $O(n)$ geometric objects, supporting some form of update and query (a query is any operation which never impacts the result of any subsequent operation). We say that a data structure (or that the set of objects it maintains) is *incremental* when it allows updates which consist of inserting a new object in S . We use the term *fully-dynamic* when both insertions and deletions are allowed. The set S can be initialized in a preprocessing phase.

2.1 General reduction schemes

All our reductions have the same basic structure based on a geometric view of Pătraşcu’s Multiphase problem [50], where we encode a family $\mathcal{F} = \{F_1, \dots, F_k\}$ of subsets of $\{1, \dots, m\}$ as a grid of objects where the presence (or absence) of an object at the grid coordinates (x, y) encodes $x \in F_y$. We can then select some of the columns $I \in \{1, \dots, k\}$ and a row $j \in \{1, \dots, m\}$, allowing us to test if $I \cap F_j \neq \emptyset$ efficiently. We abstract some of the commonalities of the reductions in the following “general” reduction schemes, so we can focus on the specifics of each problem and avoid repetitions later on. Rather than give the original definition of the Multiphase problem, let us define what it means for a data structure to solve it, as this will make the statements of reductions easier, more uniform, and makes the required constraints on the data structure we consider explicit.

► **Definition 5** (Solving the Multiphase problem). *Let $\mathcal{F} = \{F_1, \dots, F_k\}$ be a family of k subsets of $\{1, 2, \dots, m\}$. Let $s_{\mathcal{F}} = \sum_{F \in \mathcal{F}} |F|$. Consider a data structure D with an undo operation⁴ which maintains a set S of $O(n)$ objects with expected preprocessing time $O(t_p)$, expected amortized update time $O(t_u)$ and expected amortized query time $O(t_q)$. Suppose it allows us to do the following.*

- (Step 1) *First, we read \mathcal{F} and store a set of n objects in S using only the preprocessing operation of D .*
- (Step 2) *Then, we receive a subset $J \subset \{1, 2, \dots, m\}$ and perform u_J updates on S .*
- (Step 3) *Finally, we are given an index $1 \leq i \leq k$ and after $O(1)$ updates and queries on S we decide if $J \cap F_i \neq \emptyset$.*

Assume that the time of each of these three steps is dominated by the time of the operations on D and that in each step, the only information available from the previous steps is what is accessible through D . Let $t_{uq} = t_q$ if only queries are performed in Step 3, otherwise let $t_{uq} = t_u + t_q$.

We say that such a data structure solves the Multiphase problem.

As mentioned in the introduction, Pătraşcu gave lower bounds on the time required to solve the Multiphase problem conditioned on the 3SUM conjecture and reduced this problem to various dynamic problems. His reduction from 3SUM has since been tightened by Kopelowitz et al. [41] and reductions from the Exact Triangle and OMv conjectures have been found by Vassilevska Williams and Xu [56] and Henzinger et al. [34] respectively.

We summarize the implications from these works for different parameters in the following theorems. While this results in somewhat verbose statements, we chose this approach in order to streamline the reductions in this paper and to make the lower bounds we obtain explicit in terms n .

► **Theorem 6.** *Let D be a data structure which solves the Multiphase problem. If the Exact Triangle conjecture is true (or in particular if either the 3SUM or APSP conjecture is true), then for any $0 < \gamma < 1$:*

- (Scenario 1) *If $n = O(m \cdot k)$ and $u_J = O(m)$, we have*

$$t_p + t_u \cdot n + t_{uq} \cdot n^{\frac{1+\gamma}{3-2\gamma}} = \Omega\left(n^{\frac{2}{3-2\gamma}-o(1)}\right).$$

- (Scenario 2) *If $n = O(m \cdot k)$ and $u_J = O(|J|)$, we have*

$$t_p + t_u \cdot n^{\frac{2-\gamma}{3-2\gamma}} + t_{uq} \cdot n^{\frac{1+\gamma}{3-2\gamma}} = \Omega\left(n^{\frac{2}{3-2\gamma}-o(1)}\right).$$

- (Scenario 3) *If $n = O(s_{\mathcal{F}})$ and $u_J = O(m)$, we have*

$$t_p + t_u \cdot n^{\frac{3-2\gamma}{2-\gamma}} + t_{uq} \cdot n^{\frac{1+\gamma}{2-\gamma}} = \Omega\left(n^{\frac{2}{2-\gamma}-o(1)}\right).$$

- (Scenario 4) *If $n = O(s_{\mathcal{F}})$ and $u_J = O(|J|)$, we have*

$$t_p + t_u \cdot n + t_{uq} \cdot n^{\frac{1+\gamma}{2-\gamma}} = \Omega\left(n^{\frac{2}{2-\gamma}-o(1)}\right).$$

Note that for incremental (or fully-dynamic) data structures where we can insert objects, we can always assume $t_p = O(t_u \cdot n)$ by inserting the $O(n)$ initial objects individually.

⁴ A data structure is said to have an undo operation if for any update U there is complementary update U' so that if U and U' are executed sequentially the results of subsequent operations are identical to the case where U and U' were never executed. This requirement is easily satisfied in structures that maintain a set and have insertion and deletion update operations.

The results of Henzinger et al. [34] imply that whenever we have such lower bounds from the hardness of Exact Triangle, we can get stronger bounds if we assume hardness of the OMv problem instead.

► **Theorem 7.** *Let D be a data structure which solves the Multiphase problem. Assume $n = O(m^{c_1} \cdot k^{c_2})$ for some constants $c_1, c_2 > 0$, $u_J = O(m)$, and the expected preprocessing time t_p is at most polynomial in n . If the OMv conjecture is true, then for any $0 < \gamma < 1$,*

$$t_u \cdot n^\gamma + t_{uq} \cdot n^{\frac{1-c_1\gamma}{c_2}} = \Omega\left(n^{\frac{1+(c_2-c_1)\gamma}{c_2}-o(1)}\right).$$

In particular if $n = O(m \cdot k)$ (as is the case in the four scenarios of Theorem 6), for any $0 < \gamma < 1$ we have $t_u \cdot n^\gamma + t_{uq} \cdot n^{1-\gamma} = \Omega(n^{1-o(1)})$. For $\gamma = 1/2$, we thus have $t_u + t_q = \Omega(n^{1/2-o(1)})$.

These results follow from straightforward adaptations of Pătraşcu’s proofs [50] together with the more recent results from Williams and Xu [56] and Henzinger et al. [34], and are implicit in the two latter papers. To apply these theorems, we need data structures with an undo operation. When considering structures in the fully-dynamic setting where updates consist of inserting or deleting an object, then this requirement is automatically satisfied. For structures with guarantees on the runtime per operation (rather than amortized guarantees), we can use the following standard technique (see for example [47, Theorem 2.1]).

► **Lemma 8.** *Any data structure with guarantees on the runtime per operation (non-amortized) can be augmented to support an undo operation with the same guarantees.*

From now on, whenever we consider a structure with per-operation runtime guarantees, we assume (without loss of generality) that it has been augmented to support undo.

2.2 An example: Square Range Marking

We illustrate the use of these theorems on the following problem.

Square Range Marking. Preprocess a static set of n initially unmarked points, where an update consists of marking all points in a given axis-aligned square range and a query returns if there is any unmarked point in the set.

Here the dynamic part of the problem is rather limited as only the markings of the points can change after an update, the set of points itself is static. The updates are even monotone in the sense that once a point has been marked it is never unmarked (in particular, the number of unmarked points can never increase). Even for this seemingly simple problem, we can use Theorems 6 and 7 to get the following (conditional) polynomial lower bounds.

► **Theorem 9.** *Let D be a data structure for Square Range Marking with t_p expected preprocessing time and t_u expected time per update (i.e. non-amortized). If the Exact Triangle conjecture holds, then*

$$t_p + t_u \cdot \left(n^{\frac{3-2\gamma}{2-\gamma}} + n^{\frac{1+\gamma}{2-\gamma}}\right) + t_q \cdot n^{\frac{1+\gamma}{2-\gamma}} = \Omega\left(n^{\frac{2}{2-\gamma}-o(1)}\right).$$

If the OMv conjecture holds and t_p is at most polynomial then for any $0 < \gamma < 1$

$$t_u \cdot (n^{1-\gamma} + n^\gamma) + t_q \cdot n^{1-\gamma} = \Omega\left(n^{1-o(1)}\right).$$

In particular, by setting $\gamma = 1/2$, we have $t_u + t_q = \Omega(n^{1/2-o(1)})$.

Proof. It suffices to show that such a data structure fits the conditions of Scenario 3 in Theorem 6. Let $\mathcal{F} = \{F_1, \dots, F_k\}$ be a family of k subsets of $\{1, 2, \dots, m\}$.

We perform Step 1 by initializing D with the following points: for each $1 \leq i \leq k$ and $1 \leq j \leq m$ for which $j \in F_i$, we put a point $p_{i,j}$ at coordinates $((k+2)j+1, i+1)$. The total number of points is $n = s_{\mathcal{F}}$.

To perform Step 2 when given $J \subset \{1, 2, \dots, m\}$, we mark the points inside a square range of side-length $k+2$ whose lower-left corner has coordinates $((k+2)j, 1)$, for all $j \notin J$. This requires $O(m)$ updates on D . The unmarked points are exactly the $p_{i,j}$'s such that $j \in J$.

In Step 3, when given an index $1 \leq i' \leq k$, we mark the points inside the two squares of side-length $(k+2)m$ whose lower-left corners lie at coordinates $(0, i'+1/2)$ and $(0, i'-(k+2)m-1/2)$ respectively. Now there is an unmarked point if and only if there is some point $p_{i,j}$ such that $j \in J$ and $p_{i,j}$ was not marked by these two last updates. This is the case if and only if $i = i'$. By construction, such a point exists if and only if there is some $j \in J$ such that $j \in F_{i'}$ (i.e. $J \cap F_{i'} \neq \emptyset$). Thus, we can answer a Step 3 query after two more updates to D .

By applying Theorems 6 and 7 we get the result. \blacktriangleleft

If we assume truly subquadratic expected preprocessing time we get polynomial lower bounds on t_u or t_q from the Exact Triangle conjecture. The bounds from the OMv conjecture are almost tight, as an upper bound can easily be obtained by taking a two-dimensional kd-tree [10] and augmenting it by adding markers to the nodes indicating if the points in the corresponding subtrees are marked. We then get a data structure with $O(n \log n)$ worst-case preprocessing time and $O(\sqrt{n})$ worst-case time per update. As noted by Cardinal et al. [15], using standard dynamization techniques such a data structure can even be made to support insertion and deletion of points in $O(\log^2 n)$ worst-case time.

3 Counting the number of extremal points in \mathbb{R}^3

We show polynomial lower bounds for the following problem, which in the plane can be solved in polylog worst-case time per operation by known techniques [48].

Counting Extremal Points in \mathbb{R}^3 . Maintain a dynamic set of $O(n)$ points in \mathbb{R}^3 and allow for queries counting the number of extremal points in the set.

► **Theorem 10.** *Let D be a fully-dynamic data structure for Extremal Points in \mathbb{R}^3 with t_p expected preprocessing time, t_u expected amortized update time and t_q expected amortized query time. If the Exact Triangle conjecture holds, then*

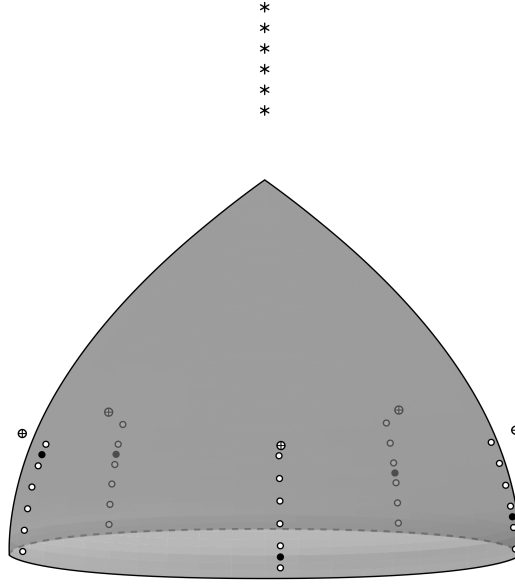
$$t_p + t_u \cdot \left(n^{\frac{2-\gamma}{3-2\gamma}} + n^{\frac{1+\gamma}{3-2\gamma}} \right) + t_q \cdot n^{\frac{1+\gamma}{3-2\gamma}} = \Omega \left(n^{\frac{2}{3-2\gamma}-o(1)} \right).$$

If the OMv conjecture holds, then for any $0 < \gamma < 1$, $t_u \cdot (n^{1-\gamma} + n^\gamma) + t_q \cdot n^{1-\gamma} = \Omega(n^{1-o(1)})$. In particular, by setting $\gamma = 1/2$, we have $t_u + t_q = \Omega(n^{1/2-o(1)})$.

Because we can assume $t_p = O(t_u \cdot n)$ this also implies that under the Exact Triangle conjecture we have $t_u = \Omega(n^{1/5-o(1)})$.

Before proving this theorem, let us introduce some notation. We let $\mathcal{F} = \{F_1, \dots, F_k\}$ be a family of k subsets of $\{1, \dots, m\}$, where $m, k \geq 5$. Here we work in cylindrical coordinates (r, θ, z) (where this would denote the point $(r \cos \theta, r \sin \theta, z)$ in Cartesian coordinates).

Let $R = 4k^2m^2$. For all $0 \leq i \leq k$ and $1 \leq j \leq m$, let $q_{i,j}$ be the point with cylindrical coordinates $(R - (2i+1)^2, \frac{2\pi}{m}j, 2i+1)$. Similarly, for all $1 \leq j \leq m$ and $1 \leq i \leq k$ such that $j \in F_i$, let $p_{i,j}$ be the point with cylindrical coordinates $(R - (2i)^2, \frac{2\pi}{m}j, 2i)$. We let $S_{\mathcal{F}}$ denote the set consisting of all these points.



■ **Figure 1** Illustration (not to scale) of the set of points obtained for $m = k = 5$ and the family of sets $\mathcal{F} = \{\{1\}, \{2\}, \dots, \{5\}\}$. The points $q_{\bullet, \bullet}$ are represented in white, the points $p_{\bullet, \bullet}$ in black and all these points lie on the translucent gray surface. The points b_{\bullet} are represented with a cross in a white circle and are above the translucent gray surface. The asterisks represent the points of the form t_{\bullet} and lie on the axis of rotational symmetry of the translucent gray surface.

For all $1 \leq j \leq m$, we let b_j denote the point with cylindrical coordinates $(R-1, \frac{2\pi}{m}j, 2k+2)$. For all $1 \leq i \leq k+1$, we let t_i denote the point lying on the longitudinal axis (the z -axis) at height $\frac{R+(2i-1)^2}{2(2i-1)}$. Note that for all $1 \leq i_1 \leq i_2 \leq k+1$, the point t_{i_1} is higher on the z -axis than t_{i_2} and that all points t_{\bullet} have a larger z -coordinate than all other previously defined points. See Figure 1 for an illustration.

► **Lemma 11.** *Let S be a set of points such that $S_{\mathcal{F}} \subset S \subset S_{\mathcal{F}} \cup \{b_j \mid 1 \leq j \leq m\} \cup \{t_i \mid 1 \leq i \leq k\}$. Let $1 \leq j' \leq m$ and $1 \leq i' \leq k$. Then:*

- *The point $q_{0, j'}$ is extremal.*
- *The point $q_{i', j'}$ is extremal if and only if $b_{j'} \notin S$ and for all $1 \leq i \leq i'$, $t_i \notin S$. If $p_{i', j'} \in S$ (i.e. $j' \in F_{i'}$), then the same holds for $p_{i', j'}$.*
- *If $t_{i'} \in S$, then $t_{i'}$ is extremal if and only if for all $1 \leq i < i'$, $t_i \notin S$.*

Moreover, this remains true even if all points are arbitrarily perturbed by moving them a distance of at most $1/R^2$.

See the full version of the paper [28] for the proof of this lemma.

Proof of Theorem 10. It suffices to show that such a data structure fits the conditions of Scenario 2 in Theorem 6. Let $\mathcal{F} = \{F_1, \dots, F_k\}$ be a family of k subsets of $\{1, 2, \dots, m\}$. We use the notation of Lemma 11. We first describe the procedure without discussing issues of finite precision and later show how this can be carried out on a Word RAM machine with words of $O(\log n)$ bits.

We perform Step 1 by initializing D with all points of the form $q_{\bullet, \bullet}$, $p_{\bullet, \bullet}$ and b_{\bullet} . This costs t_u expected time, for a total number of points $n = \Theta(m \cdot k)$.

To perform Step 2 when given $J \subset \{1, 2, \dots, m\}$, we delete the points b_j for all $j \in J$. This requires $O(|J|)$ updates on D .

In Step 3, when given an index $1 \leq i' \leq k$ we start by inserting the point $t_{i'+1}$ to D and getting the count c of extremal points. By Lemma 11, the extremal points of S at this point are exactly those of the following 6 types:

1. the point $t_{i'+1}$,
2. the points b_j for all j such that $j \notin J$,
3. the points $q_{i,j}$ for all i, j such that $j \in J$ and $i < i'$,
4. the points $p_{i,j}$ for all i, j such that $j \in F_i$, $j \in J$ and $i < i'$.
5. the points $q_{i',j}$ for all j such that $j \in J$,
6. the points $p_{i',j}$ for all j such that $j \in F_{i'}$ and $j \in J$.

To answer the query, we want to know if the number of points of the last type is greater than 0. We know that the number of points of the fifth category is exactly $|J|$. Notice that if we now insert $t_{i'}$ to D and get the new count c' of extremal points, we are counting exactly the first four categories of points, where we have replaced $t_{i'+1}$ with $t_{i'}$. Thus, we can test if the number of points of the last category is 0 simply by testing if $c - c' = |J|$. We can thus perform Step 3 with $O(1)$ updates.

We can adapt this to work on a Word RAM machine with words of length $w \geq \log n$ by moving the points to vertices of the integer lattice (after appropriate scaling). This is detailed in the full version of the paper [28]. By applying Theorems 6 and 7 we get the result. \blacktriangleleft

In the fully-dynamic setting, Chan [18] gives a data structure for this problem with $O(n^{1+\varepsilon})$ preprocessing time and $O(n^{11/12+\varepsilon})$ amortized update and query time, for an arbitrary $\varepsilon > 0$. In the more restricted semi-online setting (which generalizes the incremental case), another paper by the same author [16] gives a data structure with $O(n^{1+\varepsilon})$ preprocessing time and $O(n^{7/8+\varepsilon})$ worst-case time per operation.

4 Dynamic geometric Set Cover with squares

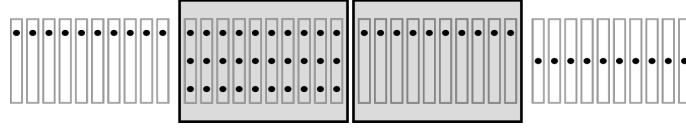
In this section we answer a question by Chan et al. [20], by giving a conditional polynomial lower bound on the time required to approximately maintain (the size of) a dynamic square set cover in the plane under range updates.

Dynamic Square Set Cover. Maintain a set S of n points and axis-aligned squares in the plane to support queries asking for the size of the smallest subset of squares which covers all points.

Even the static version of this problem with unit squares is NP-complete [29], thus the focus on approximations. Chan et al. [20] recently gave a $O(1)$ -approximate solution in the fully dynamic case where both squares and points may be inserted or deleted. This (Monte Carlo randomized) solution achieves $O(n^{1/2+\varepsilon})$ amortized update and query time. The authors ask if there is a conditional polynomial lower bound for this problem. We show the following.

► **Theorem 12.** *Let $0 \leq \alpha < 1$ be an efficiently computable⁵ constant. If there is a fully-dynamic data structure for $O(n^\alpha)$ -approximate Dynamic Square Set Cover with t_u expected amortized update time and t_q expected query time, then the Multiphase problem can be solved with $n = O\left(k^{1/(1-\alpha)} \cdot m^{2/(1-\alpha)^2}\right)$, $t_{uq} = t_u + t_q$ and $u, J = O(m)$.*

⁵ We say that a number α is efficiently computable if there is an algorithm which, for any $k \geq 0$, can output the first k bits of α in $O(\text{poly}(k))$ time.



■ **Figure 2** Illustration (not to scale) of the instance obtained after Step 2 in the proof of Theorem 12, for a data structure with a constant approximation ratio of $\beta = 3/2$. The illustrated instance has $m = 4$, $k = 3$, $\mathcal{F} = \{\{1, 2, 3\}, \{2, 4\}, \{2\}\}$, $J = \{1, 4\}$ and $c = 10$.

Together with Theorem 7 this implies polynomial lower bounds under the OMv conjecture, for all $0 \leq \alpha < 1$. In particular, for $\alpha = 0$ (i.e. for a constant approximation factor) it implies $t_u + t_q = \Omega(n^{1/3 - o(1)})$.

Proof. We first prove the result in the case of arbitrary axis-aligned rectangles and then show how to adapt it to use only squares. Let $\mathcal{F} = \{F_1, \dots, F_k\}$ be a family of k subsets of $\{1, 2, \dots, m\}$. Suppose D achieves an approximation ratio of at most $\beta \cdot n^\alpha$ for some constant integer $\beta > 1$ (we can assume this without loss of generality). Let c be an integer value which we specify later.

We perform Step 1 by initializing D with the following points and rectangles:

- for each $1 \leq i \leq k$ and $1 \leq j \leq m$ for which $j \in F_i$, we put a point $p_{i,j}^a$ at coordinates $(c \cdot j + a, i)$ for each $0 \leq a < c$;
- for each $1 \leq j \leq m$ and each $0 \leq a < c$, we put a thin vertical rectangle which covers exactly the points of the form $p_{\bullet,j}^a$.

The total number of points and rectangles at this point is $n_1 = s_{\mathcal{F}} + m \cdot c \leq m \cdot (k + 1) \cdot c$. Set c to be the smallest integer such that $c > \beta(n_1 + m + 2)^\alpha \cdot (m + 2)$ (note that the value of n_1 depends on c , but c is nonetheless well defined and can be computed in $O(\text{polylog}(m \cdot k))$ time). We then have $n_1 = O(k^{1/(1-\alpha)} \cdot m^{2/(1-\alpha)^2})$.

To perform Step 2 when given $J \subset \{1, 2, \dots, m\}$, we insert a rectangle covering all points of the form $p_{\bullet,j}^a$ for each $j \notin J$. This requires $O(m)$ updates on D . See Figure 2 for an illustration of the first two steps.

In Step 3, when given an index $1 \leq i' \leq k$, we insert two rectangles: the first covers all points $p_{i,\bullet}^a$ with $i < i'$ and the second covers all points $p_{i,\bullet}^a$ with $i > i'$. The total number of points and rectangles at this point is $n \leq n_1 + m + 2$. Now, if $J \cap F_{i'} = \emptyset$ then all points can be covered with at most $m + 2$ rectangles: the (at most) m rectangles inserted in step 2 together with the two rectangles inserted in Step 3. On the other hand, if there is some $j \in J \cap F_{i'}$, then the points of the form $p_{i,j}^a$ can only be covered by choosing c thin rectangles created in Step 1. Thus, any approximation of the set cover with a ratio better than $\frac{c}{m+2}$ suffices to distinguish between the two cases. Moreover, we have $\frac{c}{m+2} > \beta \cdot (n_1 + m + 2)^{\alpha \frac{m+2}{m+2}} \geq \beta \cdot n^\alpha$. We can then answer a Step 3 query by asking the data structure for a $\beta \cdot n^\alpha$ approximation of the size of the minimum set cover.

To see how to adapt this reduction using only squares, notice that we can increase the height of any rectangle without affecting the results. Thus, we can stretch the whole configuration of points and rectangles horizontally until the thinnest vertical rectangles become squares, and adjust the heights of the other rectangles to make them squares as well. ◀

Contrast this lower bound with the case of *unit* axis-aligned squares, for which Chan et al. [20] give a data structure for $O(1)$ -approximation achieving $2^{O(\sqrt{\log n})}$ amortized update and query time.

References

- 1 Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. 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 477–486. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.58.
- 2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.
- 3 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. doi:10.1137/15M1050987.
- 4 Pankaj K. Agarwal, Sathish Govindarajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In Rolf H. Möhring and Rajeev Raman, editors, *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, volume 2461 of *Lecture Notes in Computer Science*, pages 17–28. Springer, 2002. doi:10.1007/3-540-45749-6_6.
- 5 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 41:1–41:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.41.
- 6 Amihod 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.
- 7 Amihod Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B. Riva Shalom. Mind the gap! - online dictionary matching with one gap. *Algorithmica*, 81(6):2123–2157, 2019. doi:10.1007/s00453-018-0526-2.
- 8 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.
- 9 Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in undirected graphs: breaking the $o(m)$ barrier. 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 730–739. SIAM, 2016. doi:10.1137/1.9781611974331.ch52.
- 10 Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. doi:10.1145/361002.361007.
- 11 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 303–318. ACM, 2017. doi:10.1145/3034786.3034789.
- 12 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs under updates and in the presence of integrity constraints. In Benny Kimelfeld and Yael Amerdamer, editors, *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, volume 98 of *LIPICs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICDT.2018.8.

- 13 Karl Bringmann. Fine-grained complexity theory (tutorial). In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPICs*, pages 4:1–4:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.STACS.2019.4.
- 14 Karl Bringmann. Fine-grained complexity theory: Conditional lower bounds for computational geometry. In Liesbeth De Mol, Andreas Weiermann, Florin Manea, and David Fernández-Duque, editors, *Connecting with Computability - 17th Conference on Computability in Europe, CiE 2021, Virtual Event, Ghent, July 5-9, 2021, Proceedings*, volume 12813 of *Lecture Notes in Computer Science*, pages 60–70. Springer, 2021. doi:10.1007/978-3-030-80049-9_6.
- 15 Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Worst-case efficient dynamic geometric independent set. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPICs*, pages 25:1–25:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ESA.2021.25.
- 16 Timothy M. Chan. Semi-online maintenance of geometric optima and measures. *SIAM J. Comput.*, 32(3):700–716, 2003. doi:10.1137/S0097539702404389.
- 17 Timothy M. Chan. A (slightly) faster algorithm for Klee’s measure problem. *Computational Geometry*, 43(3):243–250, 2010. Special Issue on 24th Annual Symposium on Computational Geometry (SoCG’08). doi:10.1016/j.comgeo.2009.01.007.
- 18 Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discret. Comput. Geom.*, 64(4):1235–1252, 2020. doi:10.1007/s00454-020-00229-5.
- 19 Timothy M. Chan. More logarithmic-factor speedups for 3SUM, (median, +)-convolution, and some geometric 3SUM-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020. doi:10.1145/3363541.
- 20 Timothy M. Chan, Qizheng He, Subhash Suri, and Jie Xue. Dynamic geometric set cover, revisited. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3496–3528, 2022. doi:10.1137/1.9781611977073.139.
- 21 Timothy M. Chan and Zhengcheng Huang. Dynamic colored orthogonal range searching. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPICs*, pages 28:1–28:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ESA.2021.28.
- 22 Timothy M. Chan and Yakov Nekrich. Better data structures for colored orthogonal range reporting. 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 627–636. SIAM, 2020. doi:10.1137/1.9781611975994.38.
- 23 Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. Hardness for triangle problems under even more believable hypotheses: reductions from real APSP, real 3SUM, and OV. In Stefano Leonardi and Anupam Gupta, editors, *STOC ’22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1501–1514. ACM, 2022. doi:10.1145/3519935.3520032.
- 24 Timothy M. Chan and Da Wei Zheng. Hopcroft’s problem, log-star shaving, 2d fractional cascading, and decision trees. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 190–210, 2022. doi:10.1137/1.9781611977073.10.
- 25 Lijie Chen, Erik D. Demaine, Yuzhou Gu, Virginia Vassilevska Williams, Yinzhan Xu, and Yuancheng Yu. Nearly optimal separation between partially and fully retroactive data structures. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018, June 18-20, 2018, Malmö, Sweden*, volume 101 of *LIPICs*, pages 33:1–33:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.SWAT.2018.33.
- 26 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>.

- 27 Søren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 48:1–48:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.48.
- 28 Justin Dallant and John Iacono. Conditional lower bounds for dynamic geometric measure problems. *CoRR*, abs/2112.10095, 2022. arXiv:2112.10095.
- 29 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981. doi:10.1016/0020-0190(81)90111-3.
- 30 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.
- 31 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 621–630. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.72.
- 32 Prosenjit Gupta, Ravi Janardan, Saladi Rahul, and Michiel H. M. Smid. Computational geometry: Generalized (or colored) intersection searching. In Dinesh P. Mehta and Sartaj Sahni, editors, *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>.
- 33 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.
- 34 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.
- 35 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 26:1–26:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ITCS.2017.26.
- 36 Ravi Janardan and Mario Alberto López. Generalized intersection searching problems. *Int. J. Comput. Geom. Appl.*, 3(1):39–69, 1993. doi:10.1142/S021819599300004X.
- 37 Ce Jin and Yinzhan Xu. Tight dynamic problem lower bounds from generalized BMM and OMv. *To appear in STOC'22*. *CoRR*, abs/2202.11250, 2022. arXiv:2202.11250.
- 38 Adam Karczmarz and Jakub Lacki. Fast and simple connectivity in graph timelines. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 458–469. Springer, 2015. doi:10.1007/978-3-319-21840-3_38.
- 39 Young Kun Ko and Min Jae Song. Hardness of approximate nearest neighbor search under l -infinity. *CoRR*, abs/2011.06135, 2020. arXiv:2011.06135.
- 40 Tsvi Kopelowitz and Robert Krauthgamer. Color-distance oracles and snippets. 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 24:1–24:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.24.

- 41 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.
- 42 Kasper Green Larsen and Freek van Walderveen. Near-optimal range reporting structures for categorical data. 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 265–276. SIAM, 2013. doi:10.1137/1.9781611973105.20.
- 43 Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. 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 2182–2189. SIAM, 2017. doi:10.1137/1.9781611974782.142.
- 44 Joshua Lau and Angus Ritossa. Algorithms and hardness for multidimensional range updates and queries. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPICs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ITCS.2021.35.
- 45 Christian W. Mortensen. Generalized static orthogonal range searching in less space. Technical Report TR-2003-33, IT University of Copenhagen, Copenhagen, Denmark, September 2003.
- 46 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39(1):9:1–9:21, 2014. doi:10.1145/2543924.
- 47 Mark H. Overmars. Searching in the past II: General transforms. Technical Report RUU-CS-81-9, Department of Computer Science, University of Utrecht, Utrecht, The Netherlands, May 1981.
- 48 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981. doi:10.1016/0022-0000(81)90012-X.
- 49 Maximilian Probst. On the complexity of the (approximate) nearest colored node problem. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPICs*, pages 68:1–68:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.68.
- 50 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing, STOC '10*, pages 603–610, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1806689.1806772.
- 51 Aviad Rubinfeld. Hardness of approximate nearest neighbor search. 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 1260–1268. ACM, 2018. doi:10.1145/3188745.3188916.
- 52 Qingmin Shi and Joseph F. 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.
- 53 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 456–480. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00036.
- 54 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.
- 55 Virginia Vassilevska Williams. *On some fine-grained questions in algorithms and complexity*, pages 3447–3487. WORLD SCIENTIFIC, 2019. doi:10.1142/9789813272880_0188.

- 56 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and APSP. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 786–797. IEEE, 2020. doi:10.1109/FOCS46700.2020.00078.
- 57 Hakan Yıldız, Luca Foschini, John Hershberger, and Subhash Suri. The union of probabilistic boxes: Maintaining the volume. 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 591–602. Springer, 2011. doi:10.1007/978-3-642-23719-5_50.
- 58 Hakan Yıldız, John Hershberger, and Subhash Suri. A discrete and dynamic version of Klee’s measure problem. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry, Toronto, Ontario, Canada, August 10-12, 2011*, 2011. URL: <http://www.cccg.ca/proceedings/2011/papers/paper28.pdf>.

A Simpler QPTAS for Scheduling Jobs with Precedence Constraints

Syamantak Das ✉

Department of Comp Science and Engineering, IIT Delhi, India

Andreas Wiese ✉

Department of Mathematics, Technical University of Munich, Germany

Abstract

We study the classical scheduling problem of minimizing the makespan of a set of unit size jobs with precedence constraints on parallel identical machines. Research on the problem dates back to the landmark paper by Graham from 1966 who showed that the simple List Scheduling algorithm is a $(2 - \frac{1}{m})$ -approximation. Interestingly, it is open whether the problem is NP-hard if $m = 3$ which is one of the few remaining open problems in the seminal book by Garey and Johnson. Recently, quite some progress has been made for the setting that m is a constant. In a breakthrough paper, Levey and Rothvoss presented a $(1 + \epsilon)$ -approximation with a running time of $n^{(\log n)^{O((m^2/\epsilon^2) \log \log n)}}$ [STOC 2016, SICOMP 2019] and this running time was improved to quasi-polynomial by Garg [ICALP 2018] and to even $n^{O_{m,\epsilon}(\log^3 \log n)}$ by Li [SODA 2021]. These results use techniques like LP-hierarchies, conditioning on certain well-selected jobs, and abstractions like (partial) dyadic systems and virtually valid schedules.

In this paper, we present a QPTAS for the problem which is arguably simpler than the previous algorithms. We just guess the positions of certain jobs in the optimal solution, recurse on a set of guessed subintervals, and fill in the remaining jobs with greedy routines. We believe that also our analysis is more accessible, in particular since we do not use (LP-)hierarchies or abstractions of the problem like the ones above, but we guess properties of the optimal solution directly.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases makespan minimization, precedence constraints, QPTAS

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.40

1 Introduction

A classical problem in scheduling theory is the problem to schedule jobs on parallel machines in order to minimize the makespan, while obeying precedence constraints between the jobs. It goes back to the 1966 when Graham proved in his seminal paper [5] that the simple List Scheduling algorithm yields a $(2 - \frac{1}{m})$ -approximation algorithm. Formally, the input consists of a set J of n jobs, a number of machines $m \in \mathbb{N}$, and each job $j \in J$ is characterized by a processing time $p_j \in \mathbb{N}$. We seek to schedule them non-preemptively on m machines in order to minimize the time when the last job finishes, i.e., to minimize the makespan. Additionally, there is a precedence order \prec which is a partial order between the jobs. Whenever $j \prec j'$ for two jobs $j, j' \in J$ then job j' can only be started when j has already finished. Given that the List Scheduling algorithm is essentially a simple greedy routine, one may imagine that one can achieve a better approximation ratio with more sophisticated algorithmic techniques. However, Svensson showed that even for unit size jobs (i.e., $p_j = 1$ for each $j \in J$) there can be no $(2 - \epsilon)$ -approximation algorithm for any $\epsilon > 0$ [9], assuming a variant of the Unique Games Conjecture. Hence, under this conjecture List Scheduling is the essentially best possible algorithm. Slight improvements are known for unit size jobs: there is an algorithm by Coffman and Graham [1] which computes an optimal solution when $m = 2$, and a which is a $(2 - \frac{2}{m})$ -approximation algorithm for general m , as shown by Lam and Sethi [6]. Also, there is an $(2 - \frac{7}{3m+1})$ -approximation algorithm known



© Syamantak Das and Andreas Wiese;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 40; pp. 40:1–40:11

Leibniz International Proceedings in Informatics



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

due to Gangal and Ranade [2]. In fact, the setting of unit-size jobs is interesting since then the complexity of the problem stems purely from the precedence constraints and not from the processing times of the jobs (which might encode problems like PARTITION). We assume this case from now.

In practical settings m might be small, e.g., m might be the number of processors in a system, or the number of cores of a CPU. Thus, it is a natural question whether better approximation ratios are possible when m is a constant. Note that the mentioned lower bound of $2 - \epsilon$ [9] does not hold if $m = O(1)$. For $m = 2$ the mentioned algorithm by Coffman and Graham [1] computes an optimal solution; however, even if $m = 3$ it is not known whether the problem is NP-hard! In fact, it is one of the few remaining open problems in the book by Garey and Johnson [3].

In a break-through result, Levey and Rothvoss presented a $(1 + \epsilon)$ -approximation with a running time of $n^{(\log n)^{O((m^2/\epsilon^2) \log \log n)}}$ [7]. Subsequently, the running time was improved by Garg [4] to $n^{O_{m,\epsilon}(\log^{O(m^2/\epsilon^2)} n)}$ which is quasi-polynomial. Both algorithms are based on the natural-LP relaxation of the problem, (essentially) lifted by a certain number r of rounds of the Sherali-Adams hierarchy, and it can be solved in time $n^{O(r)}$. Given the optimal LP-solution x^* , they *condition* on certain variables in the support of x^* , which effectively fixes time slots for the corresponding jobs. Each conditioning operation changes not only the variable that one conditions on, but possibly also other variables in the support. After a well-chosen set of conditioning operations, they recurse into smaller subintervals and give each of them a copy of the current LP-solution (which might be different from x^* due to the conditioning operations). Intuitively, in each recursive call for some subinterval I they seek to schedule jobs that can only be scheduled during I according to the previous conditionings and the precedence constraints; they call these jobs *bottom jobs* and the other jobs *top* and *middle jobs*. The middle jobs can be discarded. For the top jobs, they first use a matching argument to show that most of the top jobs can be inserted if one can ignore the precedence constraints between top jobs. Knowing this, they insert most of the top jobs with a variation of Earliest-Deadline-First (EDF), such that all precedence constraints are satisfied; some of the top jobs are discarded in the process. The discarded jobs are later inserted in a greedy manner which is affordable since they are very few.

A different approach is used by Li [8] who improved the running time further to $n^{O_{m,\epsilon}(\log^3 \log n)}$. Instead of working with an LP, he guesses directly certain properties of the optimal solution. While in the above argumentation each conditioning step costs a factor $n^{O(1)}$ in the running time, he argues that – roughly speaking – most of the time the information he guesses is binary and hence costs only a factor of 2 in the running time. More precisely, he guesses properties of a technical abstraction based on *dyadic systems*, *partial dyadic systems*, and *virtually valid schedules*. On a high level, he shows that based on the optimal schedule one can define a corresponding dyadic system and a virtually valid schedule for it, at the cost of discarding a few jobs. His algorithm then searches for the dyadic system and a virtually valid schedule for it that discards as few jobs as possible. Then, he shows that based on them, he can construct an actually valid schedule, which discards only few additional jobs.

1.1 Our Contribution

In this paper we present a QPTAS for the makespan minimization problem with unit size jobs on a constant number of machines along with precedence constraints ($Pm|prec, p_j = 1|C_{\max}$ in the three-field notation) which is arguably simpler than the $(1 + \epsilon)$ -approximation algorithms

sketched above. We do not use an LP-formulation and in particular no LP-hierarchy or a similar approach based on conditioning on variables. Instead, we guess properties of the optimal solution directly, similarly as Li [8]. However, we do not use the reduction to dyadic systems or a similar abstraction but work with the optimal solution directly. We believe that this makes the algorithm and the analysis easier to understand. Our running time is $n^{O_{m,\epsilon}((\log n)^{m/\epsilon})}$ so it is asymptotically better than the running time by Garg [4] up to hidden constants.

Our algorithm is actually pretty simple. Let T be the optimal makespan (which we guess). For a parameter $k = (\log n)^{O(1)}$ we guess the placement of k jobs in OPT and a partition of $[0, T)$ into at most k intervals. For each interval I , there are some jobs that need to be scheduled during I according to our guesses and the precedence constraints. We recurse on each interval I and its corresponding jobs. Then, we add all remaining jobs with a simple variant of EDF where the release dates and deadlines are defined such that the precedence constraints between these jobs and the other jobs (that we recursed on) are satisfied. For the correct guesses, we show that the resulting schedule discards at most $O(\epsilon T)$ jobs, and we add those jobs at the end with a simple greedy routine.

In our analysis, we use a hierarchical decomposition of intervals, like the previous results [8, 7, 4]. In contrast to those, we define the decomposition such that each interval is subdivided into $O(\log n/\epsilon)$ subintervals (instead of 2 subintervals) since this makes the analysis easier. Based on the optimal solution, for each level we identify certain jobs that we want to guess in that level later. Also, we assign a level to each job. Via a shifting step, we show that we can discard the jobs from intuitively every (m/ϵ) -th level. Based on these levels, we argue that there are guesses for our algorithm that yield a small total number of discarded jobs. Intuitively, we guess the jobs from the first m/ϵ levels that we identified above and the intervals of the $(m/\epsilon + 1)$ -th level. We recurse on the latter intervals. When we insert the jobs of the first m/ϵ levels via EDF (those jobs that we did not guess already), the jobs from our recursion and their precedence constraints dictate for each inserted job j a time window $[r_j, d_j)$. It might happen that the position of j in the optimal solution is not contained in $[r_j, d_j)$, but we show that it is always contained in the larger time window of $[r_j - \lambda, d_j + \lambda)$ for a small value $\lambda > 0$. We show that we need to discard at most $O(mT \frac{\epsilon}{\log n})$ jobs to compensate for this error and due to the precedence constraints between the inserted jobs. We analyze EDF directly and do not need to go via a matching argumentation as done in [7]. It turns out that our algorithm recurses for at most $O(\frac{\epsilon}{m} \log n)$ levels and hence this yields $O(\frac{\epsilon \log n}{m} \cdot mT \frac{\epsilon}{\log n}) = O(\epsilon^2 T)$ discarded jobs in total.

While our analysis borrows ideas from the mentioned previous results [8, 7, 4] we believe that our algorithm and our analysis are simpler and more accessible.

2 Algorithm

We present a simple QPTAS for the problem $Pm|prec, p_j = 1|C_{\max}$. Let $\epsilon > 0$ and assume w.l.o.g. that $1/\epsilon \in \mathbb{N}$. We guess $T := \text{OPT}$ and assume w.l.o.g. that T is a power of 2 (if this is not the case, then we can add some dummy jobs that need to be processed after all other jobs; note that a $(1 + \epsilon)$ -approximate solution for this larger instance is a $(1 + 2\epsilon)$ -approximate solution for the original instance). Also, w.l.o.g. we assume that in OPT each job starts and ends at an integral time point, i.e., during a time interval of the form $[t, t + 1)$ for some $t \in \mathbb{N}$. We will refer to such a time interval as a *time slot*. Furthermore, we assume that w.l.o.g. that the precedence constraints are transitive, i.e., if $j \prec j'$ and $j' \prec j''$ then also $j \prec j''$.

Our algorithm works on a “guess and recurse” framework. Define a parameter $k := (m \log n / \varepsilon)^{m/\varepsilon+1}$. The reader can think of the parameter k as $O_{m,\varepsilon}((\log n)^{m/\varepsilon+1})$, where $O_{m,\varepsilon}()$ hides constants that are only dependent on m, ε . Our algorithm has three steps. First, we guess up to k jobs from OPT and their time slots in OPT, i.e., we try all combinations of up to k input jobs and all combinations for their time slots to schedule them. Then, we guess a partition of $[0, T)$ into at most k intervals, i.e., we try all combinations of partitioning $[0, T)$ into at most k intervals. By allowing empty intervals, we assume w.l.o.g. that we guess exactly k intervals and denote them by I_1, \dots, I_k . Let J_{guess} denote the guessed jobs. There might be precedence constraints between jobs in J_{guess} and jobs in $J \setminus J_{guess}$. In particular, these precedence constraints might dictate that some job $j \in J \setminus J_{guess}$ needs to be scheduled within some interval I_j . In this case, we say that j is a *bottom job*. Let $J_{bottom} \subseteq J \setminus J_{guess}$ denote the set of all bottom jobs. We call each job $j \in J \setminus (J_{guess} \cup J_{bottom}) =: J_{top}$ a *top job*.

Given our guess for the jobs J_{guess} and their time slots and our guessed partitioning of $[0, T)$, we make k recursive calls: one for each interval I_i with $i = 1, 2, \dots, k$. Let us denote by $J_{bottom}^{(i)}$ the subset of jobs in J_{bottom} that need to be scheduled within I_i according to our guesses above, and let $J_{guess}^{(i)}$ denote the jobs in J_{guess} for which we guessed a time slot within I_i . We make a recursive call on the interval I_i whose input are the jobs $J_{bottom}^{(i)} \cup J_{guess}^{(i)}$ and the guessed time slots for the jobs in $J_{guess}^{(i)}$. In this recursive call, we want to compute a schedule in which

- all jobs in $J_{guess}^{(i)}$ are scheduled in the time slots that we had guessed for them,
- a (hopefully very large) subset of the jobs in $J_{bottom}^{(i)}$ are scheduled; we denote by $J_{disc}^{(i)} \subseteq J_{bottom}^{(i)}$ the jobs in $J_{bottom}^{(i)}$ that were not scheduled, we call them the *discarded jobs*, and
- we obey the precedence constraints between the jobs in $J_{bottom}^{(i)} \cup J_{guess}^{(i)}$. We ignore all precedence constraints that involve the jobs in $J_{disc}^{(i)}$.

Suppose that we are given a solution from each recursive call. We define $J_{disc} := \bigcup_{i=1}^k J_{disc}^{(i)}$. We ignore these discarded jobs for now. We want to schedule the jobs in J_{top} . Recall that these are jobs in $J \setminus J_{bottom} \cup J_{guess}$. To this end, for each job $j \in J_{top}$ we define an artificial release date r_j and an artificial deadline d_j . Let $j \in J_{top}$. We define r_j to be the earliest start time of an interval I_i at which each job $j' \in J_{guess} \cup J_{bottom}$ with $j' \prec j$ has completed; we define $r_j := 0$ if there is no such job j' . Similarly, we define d_j to be the latest end time of an interval I_i at which no job $j'' \in J_{guess} \cup J_{bottom}$ with $j \prec j''$ has already started; again, if there is no such job j'' we define $d_j := T$. In order to find slots for the jobs in J_{top} we use the following variation of the Earliest Deadline First (EDF) algorithm. We sweep the time axis from left to right. For each time $t = 0, 1, 2, \dots$ we consider the not yet scheduled jobs $j \in J_{top}$ with $r_j \leq t$ whose predecessors in $J_{guess} \cup J_{bottom} \setminus J_{disc}$ we have already scheduled before time t . We sort these jobs non-decreasingly by their deadlines (breaking ties arbitrarily) and add them in this order to the machines that are idle during $[t, t+1)$. We do this until no more machine is idle during $[t, t+1)$. It might happen that a job $j \in J_{top}$ misses its deadline at the current time t , i.e., it holds that $t = d_j$ but j has not been scheduled by us at any slot $r_j \leq t' \leq t$ and during $[t', t'+1)$ all machines are busy. In this case we add j to the set J_{disc} . Among all our guesses for the jobs and the partition into intervals, we output the solution in which at the very end the smallest number of jobs is in the set J_{disc} (breaking ties arbitrarily).

Each recursive call for an interval works similarly as the main call of the recursion described above. The (straightforward) differences are the following: the input of each recursive call consists of the interval \bar{I} , a set of jobs \bar{J}_{guess} which were guessed in previous levels in the recursion, together with their guessed time slots, and a set of (not yet scheduled)

jobs \bar{J} . We guess the time slots for up to k guessed jobs $J_{guess} \subseteq \bar{J}$, and we require that these guesses do not violate the precedence constraints with the jobs in \bar{J}_{guess} . Also, we guess a partition of I into k intervals (rather than a partition of $[0, T)$). The input for each recursive call for a subinterval \bar{I}_i of \bar{I} consists of \bar{I}_i , $\bar{J}_{guess} \cup J_{guess}$, and the jobs in \bar{J} that need to be scheduled within \bar{I}_i according to our guesses for $\bar{J}_{guess} \cup J_{guess}$. We return the computed schedule for \bar{I} for a subset of the jobs $\bar{J}_{guess} \cup \bar{J}$ and the discarded jobs in $\bar{J}_{guess} \cup \bar{J}$.

If the algorithm is called on an interval of length 1 then we skip the step of partitioning the interval further into at most k subintervals (and in particular we do not recurse anymore). We will show later that there are guesses that lead to an $(1 + \epsilon)$ -approximate solution such that the recursion depth is $\frac{\epsilon}{m} \log n$. In order to enforce that the recursion depth is $\frac{\epsilon}{m} \log n$ (limiting the running time of the algorithm), we define that a recursive call at recursion depth $\frac{\epsilon}{m} \log n + 1$ simply outputs a solution in which all of the jobs in \bar{J} are discarded and the algorithm does not recurse further.

After running the recursive algorithm described above, we need to schedule the jobs in J_{disc} . Intuitively, for each such job $j \in J_{disc}$ we create an empty time slot that we add into our schedule and inside which we schedule j . This will increase our makespan by $|J_{disc}|$. Formally, we consider the jobs $j \in J_{disc}$ in an arbitrary order. For each job $j \in J_{disc}$ we determine a time t such that all its predecessors in our current schedule have finished by time t , but none of its successors in our current schedule have started yet. Such a time t always exists since we show later that we obtain a feasible schedule for all jobs in $J \setminus J_{disc}$ and we assumed that the precedence constraints are transitive. We insert an empty time slot $[t, t + 1]$ into our schedule, i.e., we move all jobs scheduled during $[t, \infty)$ by one unit to the right, and we schedule j during $[t, t + 1]$. This completes the description of our algorithm.

3 Analysis

In this section, we prove that the above algorithm is a QPTAS.

► **Lemma 1.** *The above algorithm runs in time $n^{(\log n)^{O_{m,\epsilon}(m/\epsilon)}}$.*

Proof. For the running time, we observe that there are at most $n^{O(k)}$ choices for the guessed k jobs, at most $T^{O(k)} = (n)^{O(k)} = n^{O(k)}$ choices for their time slots, and similarly $n^{O(k)}$ choices for the at most k intervals that we guess. Since we bound the recursion depth to be at most $\log n + 1$, this yields a running time of $n^{O(k \log n)} = n^{(\log n)^{O_{m,\epsilon}(m/\epsilon)}}$. ◀

Now we prove the approximation ratio of our algorithm. To this end, we define guesses for jobs and their time slots and intervals in each call of our recursion, such that for these guesses our algorithm outputs a schedule with makespan at most $1 + 6\epsilon$.

3.1 Laminar family of intervals

For this, we define a laminar family \mathcal{L} of intervals. Recall that we assumed that T , the guessed makespan, is a power of 2. We define that the entire interval $[0, T)$ forms the (only) interval of level 0. Let us define $\rho = \lceil \log(\log n / \epsilon) \rceil$. Consider an interval I of some level $\ell = 0, 1, 2, \dots$ (we will argue the number of levels later). If $|I| \geq 2^\rho$ then I is partitioned into $2^\rho = \Theta(\log n / \epsilon)$ equal-sized intervals of length $|I|/2^\rho$. These intervals constitute the level $\ell + 1$ of the family \mathcal{L} . For each interval $I \in \mathcal{L}$, we denote by $\ell(I)$ the level of I .

If $1 < |I| < \rho$ then I is partitioned into $|I|$ intervals of level $\ell + 1$ of length 1 each. If $|I| = 1$ then I is not partitioned further.

► **Lemma 2.** *The total number of levels in the laminar family \mathcal{L} is at most $\log n / \log(\log n / \varepsilon) + 1$.*

Proof. By construction, each interval at a particular level $\ell = 0, 1, 2, \dots$ is of equal length. Hence the length of an interval at level ℓ is at most $T / (\log n / \varepsilon)^\ell$. Further, once the length of intervals of a level becomes less than $2^p \leq 2 \log n / \varepsilon$, there could only be one additional level where every interval is of length 1. Hence the total number of levels could be at most $\log T / \log(\log n / \varepsilon) + 1 \leq \log n / \log(\log n / \varepsilon) + 1$ ◀

3.2 Gussed, top, and bottom jobs

Next, we assign the jobs to levels. More precisely, for each level ℓ we define a set of *gussed jobs* $J_{guess}^{(\ell)}$ and a set of *top jobs* $J_{top}^{(\ell)}$. The intuition is that later we want to guess the jobs in $\bigcup_{\ell'=0}^{\ell} J_{guess}^{(\ell')}$ and the jobs in $\bigcup_{\ell'=0}^{\ell} J_{top}^{(\ell')}$ will form top jobs. We say that a *chain of jobs* is a set of jobs $J' = \{j_1, j_2, \dots, j_c\}$ for some $s \in \mathbb{N}$ such that $j_i \prec j_{i+1}$ for each $i \in \{1, \dots, c-1\}$, and we say that c is the *length* of the chain.

We define these sets $J_{guess}^{(\ell)}$ and $J_{top}^{(\ell)}$ level by level in the order $\ell = 0, 1, 2, \dots$. Consider a level ℓ . Let I be an interval of level ℓ . We initialize $J_{guess}^{(\ell)} = J_{top}^{(\ell)} = \emptyset$. Our plan is that we add jobs to $J_{guess}^{(\ell)}$ step by step. Let J_I denote the jobs that can only be scheduled during I , assuming that we schedule the jobs in $J_{guess}^{(0)} \cup \dots \cup J_{guess}^{(\ell-1)}$ exactly as in OPT. We say that a job $j \in J_I$ is *flexible* if it can still be scheduled in more than one subinterval of level $\ell + 1$ of I , assuming that we schedule the jobs in $J_{guess}^{(0)} \cup \dots \cup J_{guess}^{(\ell)}$ exactly as in OPT. Suppose that there is a chain $J' \subseteq J_I \setminus J_{guess}^{(\ell)}$ of length at least $\varepsilon |I| / 2^{\lceil \log \log n \rceil}$ that contains only flexible jobs. Then for each interval I' of level $\ell + 1$ we add to $J_{guess}^{(\ell)}$ the first and the last job from J' that is scheduled during I' in OPT. If we guess these jobs in our algorithm, the effect is that each job in J' that we did *not* add to $J_{guess}^{(\ell)}$ can be scheduled only during one interval of level $\ell + 1$. Hence, one way to think of this procedure is that we push these jobs one level down. We do this operation until there is no more chain J' of length at least $\varepsilon |I| / 2^{\lceil \log \log n \rceil}$ that contains only flexible jobs. We define that $J_{top,I}^{(\ell)}$ contains all remaining flexible jobs in J_I . We do this procedure for each interval I of level ℓ , and define at the end $J_{top}^{(\ell)} := \bigcup_I J_{top,I}^{(\ell)}$.

► **Proposition 3.** *For every job $j \in J$, there exists a unique $\ell \in \{0, 1, 2, \dots, \rho\}$ such that $j \in J_{top}^{(\ell)}$.*

3.3 Few rejected jobs

With the preparation above, we will show that there are guesses of our algorithm for the gussed jobs and the intervals that lead to few discarded jobs overall, at most $O(\varepsilon T)$ many. Since the algorithm selects the guesses that lead to the minimum total number of discarded jobs, we will show that it computes a solution with at most $O(\varepsilon T)$ discarded jobs. We need some preparation for this. First, we establish that we can afford to discard all jobs in sets $J_{top}^{(a+r \cdot m/\varepsilon)}$ for $r \in \mathbb{N}_0$, for some offset a .

► **Lemma 4.** *There is an offset $a \in \{0, 1, \dots, \frac{m}{\varepsilon} - 1\}$ such that $\left| \bigcup_{r \in \mathbb{N}_0} J_{top}^{(a+r \cdot m/\varepsilon+1)} \right| \leq \varepsilon T$.*

Proof. For every $a \in \{0, 1, \dots, \frac{m}{\varepsilon} - 1\}$, we define $\mathcal{L}_a = \{\ell : \ell = (a + r \cdot m/\varepsilon + 1), r \in \mathbb{N}_0\}$ and the set $\bigcup_{\ell \in \mathcal{L}_a} J_{top}^{(\ell)}$. Now Proposition 3 implies that the resulting sets $\bigcup_{\ell \in \mathcal{L}_a} J_{top}^{(\ell)}$ are pairwise disjoint. Since T is the optimal makespan, the total number of jobs cannot exceed mT . Hence, there exists some $a \in \{0, 1, \dots, \frac{m}{\varepsilon} - 1\}$ such that $|\bigcup_{\ell \in \mathcal{L}_a} J_{top}^{(\ell)}| \leq \varepsilon T$. ◀

For the root problem of the recursion, we will show that the following guesses lead to few discarded jobs overall: the guessed subintervals are the subintervals of the laminar family at level $a + 1$ where a which is the offset as identified by the above lemma. The guessed jobs are all jobs in $\bigcup_{\ell=0}^a J_{guess}^{(\ell)}$ and we guess their time slots in OPT. We recurse on the guessed intervals of level $a + 1$ of the laminar family. Suppose that in some level $r \in \mathbb{N}$ of the recursion we are given as input an interval \bar{I} of some level $\ell_r = a + (r - 1) \cdot m/\epsilon$ of the laminar family, together with a set of jobs of the form $\bar{J} = \bar{J}_{bottom} \dot{\cup} \bar{J}_{guess}$ such that for each job $j \in \bar{J}_{guess}$ we are given a (guessed) time slot that equals the time slot in OPT during which j is executed, but we are not given a time slot for any job in \bar{J}_{bottom} . We will show that the following guesses lead to few discarded jobs overall: we guess the intervals of level $\ell_{r+1} = a + r \cdot m/\epsilon + 1$ of the laminar family; the guessed jobs are all jobs in $\bar{J}_{bottom} \cap \bigcup_{\ell=a+(r-1)\cdot m/\epsilon+1}^{a+r\cdot m/\epsilon} J_{guess}^{(\ell)}$ that are scheduled in \bar{I} in OPT and we guess their time slots in OPT.

With the next lemma, we prove inductively that there are few discarded jobs. We define $r_{\max} := \lceil \epsilon(\log n / \log \log n + 1) / m \rceil$ which is an upper bound on the number of recursion levels that we need in this way.

► **Lemma 5.** *Consider a recursive call of our algorithm in which the input is of the following form:*

- *an interval \bar{I} such that $\bar{I} = [0, T)$ (we define $r = 0$ in this case) or \bar{I} is an interval of some level $\ell_r = a + (r - 1) \cdot m/\epsilon$ of the laminar family, for some $r \in \mathbb{N}$,*
- *a set of jobs $\bar{J} = \bar{J}_{bottom} \dot{\cup} \bar{J}_{guess}$ such that*
 - *for each job $j \in \bar{J}_{guess}$ we are given a time slot that coincides with the time slot during which j is scheduled in OPT,*
 - *each job $j \in \bar{J}_{bottom}$ is scheduled during \bar{I} in OPT.*

Then our algorithm returns a schedule for \bar{J} in which at most

$$\sum_{\ell' \in \mathcal{L}_a} \sum_{\bar{I}' \in \mathcal{L}(\bar{I}') = \ell' \wedge \bar{I}' \subsetneq \bar{I}} \left| J_{top, \bar{I}'}^{(\ell')} \right| + \frac{5m\epsilon}{\log n} (r_{\max} - r) |\bar{I}| \quad (1)$$

jobs are discarded.

Our goal is now to prove Lemma 5. Consider a recursive call of our algorithm of the form specified in Lemma 5 for some $r \in \mathbb{N}$. If $r = r_{\max}$ then our algorithm simply enumerates over all possible schedules for \bar{J}_{bottom} and thus finds a schedule in which no job is discarded (since this is the case in OPT). Suppose by induction that the claim is true for all $r \geq r^* + 1$ for some r^* . We want to prove that it is true also for $r = r^*$ so we consider such a recursive call. Let \tilde{J}_{guess} denote the guessed jobs and let $\tilde{I}_1, \dots, \tilde{I}_k$ denote the guessed partition of \bar{I} into subintervals, according to our description right before Lemma 5. Let $\tilde{J}_{top} \subseteq \bar{J}$ and $\tilde{J}_{bottom} \subseteq \bar{J}$ denote the resulting set of top and bottom jobs, respectively (thus, the sets $\bar{J}, \bar{J}_{guess}, \bar{J}_{bottom}$ are part of the input, while the sets $\tilde{J}_{guess}, \tilde{J}_{top}, \tilde{J}_{bottom}$ and intervals $\tilde{I}_1, \dots, \tilde{I}_k$ stem from our guesses). Recall that for each job $j \in \tilde{J}_{top}$ we define a release time r_j and a deadline d_j in our algorithm. Let λ denote the length of each interval \tilde{I}_i (note that they all have the same length), i.e., the length of the intervals of level $\ell_{r+1} = a + r \cdot m/\epsilon + 1$ (where $r = 0$ in the root problem of the recursion). Note that $\lambda \leq |\bar{I}| / (\log n / \epsilon)^{m/\epsilon+1}$. We show in the next lemma that in OPT each job $j \in \tilde{J}_{top}$ is essentially scheduled during $[r_j, d_j]$ and thus r_j and d_j are almost consistent with OPT.

► **Lemma 6.** *For each job $j \in \tilde{J}_{top}$ it holds that in OPT the job j is scheduled during $[r_j - \lambda, d_j + \lambda)$.*

Proof. Let us recall that for each job $j \in \bar{J}_{top}$ the release time r_j is defined to be the earliest start time of an interval $\tilde{I}_i, i = 1, 2, 3, \dots, k$ such that every job $j' \in \bar{J}_{guess} \cup \bar{J}_{bottom}$ with $j' \prec j$ is completed before r_j . We want to prove in OPT the job j does not start before time $r_j - \lambda$. Let $\hat{I} = [t_1, t_2]$ denote the interval of level $\ell_{r+1} = a + r \cdot m/\epsilon + 1$ for which $t_2 = r_j$ (and observe that $t_2 = t_1 + \lambda$). By definition of r_j , there exists a job $j' \in \bar{J}_{guess} \cup \tilde{J}_{guess} \cup \tilde{J}_{bottom}$ with $j' \prec j$ that completes in \hat{I} in the schedule that we obtained from the recursive call in \hat{I} . Since we assumed that our guessed time slots for the jobs in $\bar{J}_{guess} \cup \tilde{J}_{guess}$ are identical to the corresponding time slots in OPT, we conclude that also in OPT the job j' is scheduled during \hat{I} . Thus, in OPT the job j cannot start before time $t_1 = r_j - \lambda$. An analogous argument shows that j cannot be scheduled in OPT after $d_j + \lambda$. ◀

We want to show now that our variant of EDF discards only few jobs from \tilde{J}_{top} . To this end, we partition \tilde{J}_{top} into the sets $\tilde{J}_{top,1} := \tilde{J}_{top} \cap \bigcup_{\ell=a+(r-1) \cdot m/\epsilon+1}^{a+r \cdot m/\epsilon} J_{top}^{(\ell)}$ and $\tilde{J}_{top,2} := \tilde{J}_{top} \cap J_{top}^{(a+r \cdot m/\epsilon+1)}$. We can afford to discard all jobs in $\tilde{J}_{top,2}$, see (1), but we need to bound the number of discarded jobs in $\tilde{J}_{top,1}$. For a job $j \in \tilde{J}_{top}$ it can happen that $r_j = d_j$. In this case we say that j is *degenerate*. Note that a degenerate job is always discarded.

► **Lemma 7.** *There are at most $2m\epsilon|\bar{I}|/\log n$ degenerate jobs in $\tilde{J}_{top,1}$.*

Proof. Consider any job $j \in \tilde{J}_{top,1}$. By definition, there exist two adjacent intervals $I' = [t'_1, t'_2), I'' = [t'_2, t'_3)$ such that $\ell(I') = \ell(I'') = a + r \cdot m/\epsilon$ such that j can be potentially scheduled during both the intervals as dictated by the guessed jobs \bar{J}_{guess} . Thus, if j is degenerate, then $r_j = d_j = t'_2$. Further, by Lemma 6, all jobs $j \in \tilde{J}_{top,1}, r_j = d_j = t'_2$ must be scheduled in OPT in the interval $[t'_2 - \lambda, t'_2 + \lambda]$. Hence the total number of such jobs is upper bounded by

$$\begin{aligned} & m \cdot 2\lambda |\{I' : I' \in \mathcal{L} \wedge \ell(I') = a + r \cdot m/\epsilon \wedge I' \subset \bar{I}\}| \\ & \leq 2m \cdot (\log n/\epsilon)^{m/\epsilon} \cdot \frac{|\bar{I}|}{(\log n/\epsilon)^{m/\epsilon+1}} \\ & = 2m\epsilon|\bar{I}|/\log n \end{aligned} \quad \blacktriangleleft$$

Our goal now is to bound the number of discarded non-degenerate jobs in $\tilde{J}_{top,1}$. We partition \bar{I} into *meta-intervals* $\hat{I}_1, \hat{I}_2, \dots, \hat{I}_{k'}$ with $k' \leq k$ with the properties that each interval $\hat{I}_i \in \{\hat{I}_1, \hat{I}_2, \dots\}$ is of the form $\hat{I}_i = [t_1, t_2)$ for some $t_1, t_2 \in \mathbb{N}$ such that

- each value t_1, t_2 is the start or the end point of some interval in $\tilde{I}_1, \dots, \tilde{I}_k$,
- at time t_2 there is no (non-degenerate) job $j \in \tilde{J}_{top}$ pending that was released before t_2 ,
- for each $t \in [t_1, t_2)$ such that t is the start or end point of some interval in $\tilde{I}_1, \dots, \tilde{I}_k$, some non-degenerate job $j \in \tilde{J}_{top}$ is pending at time t .

Now the intuition is that during each meta-interval $\hat{I}_i = [t_1, t_2)$ EDF tries to schedule only jobs that OPT schedules during $[t_1 - \lambda, t_2 + \lambda)$ (due to Lemma 6), so essentially we have enough space on our machines to schedule all these jobs. We might waste space due to the precedence constraints. However, this space is bounded via the following lemma.

► **Lemma 8.** *Let \tilde{I}_i be an interval such that at each time $t \in \tilde{I}_i$ some job $j \in \tilde{J}_{top}$ is pending. Then during \tilde{I}_i there are at most $|\tilde{I}_i| \frac{\epsilon}{\log n}$ time slots $[t, t+1)$ with $t \in \mathbb{N}$ such that some machine is idle during $[t, t+1)$.*

Proof. For the interval $\hat{I}_i = [t_1, t_2)$, let $J_{\hat{I}_i}$ denote the subset of jobs in $j \in \tilde{J}_{top,1}$ such that $r_j \leq t_1$. We now create a partition of $J_{\hat{I}_i}$ according to the precedence constraints. Let $J_0 \subseteq J_{\hat{I}_i}$ denote the jobs whose preceding jobs have been either scheduled or discarded

before t_1 . For every $p = 1, 2, \dots, \eta$ (where η is some positive integer), let J_p denote the set of jobs $j \in J_{\hat{I}_i}$ for which there exists a job $j' \in J_{p-1}$ such that $j' \prec j$ and there is no job $j'' \in J_{\hat{I}_i} \setminus J_{p-1}$ such that $j'' \prec j$.

Let $\tau_1, \tau_2, \tau_3, \dots, \tau_{\eta'}$ be defined such that the time slots $[\tau_1, \tau_1 + 1), [\tau_2, \tau_2 + 1), \dots, [\tau_{\eta'}, \tau_{\eta'} + 1)$ are exactly the time slots during \hat{I}_i such that some machine is idle for the entire respective time slot. We claim that any job $j' \in \tilde{J}_{top}$ that is pending at the end of a time slot $[\tau_q, \tau_q + 1)$ with $q = \{1, 2, 3, \dots\}$ must belong to J_p for some $p > q$. We prove this by induction. Since no jobs are released inside \hat{I}_i , no job in J_0 is pending at time $\tau_0 + 1$ since otherwise this would contradict the definition of our variant of EDF and the fact that a machine is idle during $[\tau_0, \tau_0 + 1)$. Now assume the hypothesis to be true for the time slots $[\tau_1, \tau_1 + 1), [\tau_2, \tau_2 + 1), \dots, [\tau_q, \tau_q + 1)$ for some q and consider the time slot $[\tau_{q+1}, \tau_{q+1} + 1)$. If a job $j' \in \tilde{J}_{top}$ is pending at time $\tau_q + 1$ then $j' \in J_{p'}$ for some $p' > q$ by the induction hypothesis. This means that all jobs in $\bigcup_{p=0}^q J_p$ have completed before time $\tau_q + 1$. Hence, the jobs in J_{q+1} can be scheduled at any time after time $\tau_q + 1$. Suppose that there is a pending job j at time $\tau_{q+1} + 1$. Since a machine is idle during $[\tau_{q+1}, \tau_{q+1} + 1)$ we have that $j \in J_{p'}$ for some $p' > q + 1$ since otherwise our variant of EDF would have scheduled j during $[\tau_{q+1}, \tau_{q+1} + 1)$. By our construction of the guessed, top and bottom jobs in Section 3.2 the length of the longest chain of the jobs in \tilde{J}_{top} is at most $\varepsilon |\tilde{I}_i| / 2^{\lceil \log \log n \rceil} \leq |\tilde{I}_i| \frac{\varepsilon}{\log n}$. Therefore, $\eta' \leq \eta \leq |\tilde{I}_i| \frac{\varepsilon}{\log n}$ which completes our proof. \blacktriangleleft

Also, we show that there are only few meta-intervals which – together with the argumentation above – yields the following lemma.

► **Lemma 9.** *The total number of discarded non-degenerate jobs in $\tilde{J}_{top,1}$ is at most $m|\bar{I}| \left(\frac{2\varepsilon}{\log^2 n} + \frac{\varepsilon}{\log n} \right)$.*

Proof. Consider a meta-interval \hat{I}_i . Let j be the last non-degenerate job in $\bar{J}_{top,1}$ that is discarded during \hat{I}_i . Let t_1 denote the beginning of the interval \hat{I}_i . We know that d_j is the end point of some interval $\tilde{I}_{i'}$. Since j is non-degenerate we know that $[r_j, d_j] \neq \emptyset$ and by definition of \hat{I}_i this implies that at each time $t \in [t_1, d_j]$ there is some job from \bar{J}_{top} pending. Using Lemma 8, the total number of (wasted) idle slots across all machines during $[t_1, d_j]$ is at most $m\varepsilon|[t_1, d_j]|/m \log n$.

We further invoke Lemma 6 to argue that all jobs in \tilde{J}_{top} that we schedule or discard during $[t_1, d_j]$ are scheduled in OPT during $[t_1 - \lambda, d_j + \lambda)$. The maximum number of jobs that OPT could have scheduled during $[t_1 - \lambda, d_j + \lambda)$ is hence $m(d_j - t_1 + 2\lambda)$. Since our wasted space during $[t_1, d_j]$ is at most $m\varepsilon|[t_1, d_j]|/m \log n$, we conclude that we discard at most

$$m(d_j - t_1 + 2\lambda) - m(d_j - t_1) + \frac{\varepsilon}{\log n}(d_j - t_1) = 2\lambda m + \frac{\varepsilon}{\log n}(d_j - t_1)$$

jobs during $[t_1, d_j]$. By definition, for each job $j \in \bar{J}_{top,1}$ we have that $j \in J_{top}^{(\ell)}$ for some level ℓ with $\ell \leq a + (r + 1) \cdot m/\varepsilon - 1$. Thus, for such a job j we have that r_j and d_j lie in different intervals of level $a + (r + 1) \cdot m/\varepsilon$ of \mathcal{L} . Thus, if during a meta-interval \hat{I}_i a job $j \in \bar{J}_{top,1}$ is discarded, then \hat{I}_i has non-empty intersection with at least two different intervals of level $a + (r + 1) \cdot m/\varepsilon$ of \mathcal{L} . Hence, there can be at most $(\log n/\varepsilon)^{m/\varepsilon-1}$ such meta-intervals. We conclude that in total we discard at most

$$(\log n/\varepsilon)^{m/\varepsilon-1} \cdot 2\lambda m + \frac{\varepsilon}{\log n} |\bar{I}| \leq m \frac{2|\bar{I}|(\log n/\varepsilon)^{m/\varepsilon-1}}{(\log n/\varepsilon)^{m/\varepsilon+1}} + \frac{\varepsilon}{\log n} |\bar{I}| \leq m|\bar{I}| \left(\frac{2\varepsilon}{\log^2 n} + \frac{\varepsilon}{\log n} \right)$$

jobs in $\bar{J}_{top,1}$. \blacktriangleleft

40:10 A Simpler QPTAS for Scheduling Jobs with Precedence Constraints

Now we are ready to bound the total number of discarded jobs using Lemmas 7, 9, and the induction hypothesis.

Proof of Lemma 5. We shall prove the lemma for the input interval \bar{I} at level $\ell_{r^*} = a + (r^* - 1) \cdot m/\varepsilon$. By induction hypothesis, suppose (1) is true for all $r \geq r^* + 1$. Hence, applying our recursive algorithm with the input intervals $\tilde{I}_i, i = 1, 2, \dots, k$ returns a schedule where number of discarded jobs is at most

$$\sum_{\ell' \in \mathcal{L}_a} \sum_{\bar{I}' \in \mathcal{L}: \ell(\bar{I}') = \ell' \wedge \bar{I}' \subsetneq \tilde{I}_i} \left| J_{top, \bar{I}'}^{(\ell')} \right| + \frac{5m\varepsilon}{\log n} (r_{\max} - (r^* + 1)) |\tilde{I}_i| \quad (2)$$

Summing (2) over all $i = 1, 2, \dots, k$ and observing that $\bar{I} = \bigcup_{i=1}^k \tilde{I}_i$ yields that the total number of discarded jobs is at most

$$\sum_{\ell' \in \mathcal{L}_a} \sum_{\bar{I}' \in \mathcal{L}: \ell(\bar{I}') = \ell' \wedge \bar{I}' \subsetneq \bar{I}} \left| J_{top, \bar{I}'}^{(\ell')} \right| + \frac{5m\varepsilon}{\log n} (r_{\max} - (r^* + 1)) |\bar{I}| \quad (3)$$

We would now like to prove the statement for $r = r^*$. Now at the recursive call at level $r = r^*$ with the input subinterval $|\bar{I}|$ consider the guesses as described in the preceding discussion. Using Lemmas 9 and 7, the total number of jobs rejected from $\tilde{J}_{top,1}$ under these guesses is at most

$$2\varepsilon m |\bar{I}| / \log n + m |\bar{I}| \left(\frac{2\varepsilon}{\log^2 n} + \frac{\varepsilon}{\log n} \right) \leq 5m\varepsilon |\bar{I}| / \log n \quad (4)$$

Further, we could have potentially rejected all the jobs in $\tilde{J}_{top,2}$. The total number of such jobs is

$$|\tilde{J}_{top,2}| = \sum_{\bar{I}' \in \mathcal{L}: \ell(\bar{I}') = \ell', \ell' = a + r^* \cdot m/\varepsilon} \left| J_{top, \bar{I}'}^{(\ell')} \right| \quad (5)$$

Adding the above two quantities (4) and (5) to the quantity (3) and observing that our recursive algorithm selects the guesses at a particular level that minimizes the number of discarded jobs, the lemma holds for a recursive call at level r^* . ◀

► **Lemma 10.** *Our algorithm computes a solution with a makespan of at most $(1 + 6\varepsilon)T$.*

Proof. The input at the top level recursive call in our algorithm is an interval $\bar{I} = [0, T)$ and the entire set of jobs J . Plugging in $r = 0$ in Lemma 5 and using Lemma 4, the first term in (1) is bounded by $\sum_{\ell' \in \mathcal{L}_a} |J_{top}^{\ell'}| \leq \varepsilon T$. Since $r_{\max} := \lceil \varepsilon(\log n / \log \log n + 1) / m \rceil$, the second term for $r = 0$ in (1) bounded by $5\varepsilon^2 T / (\log \log n + 1)$.

Hence, the number of discarded jobs in our algorithm is upper bounded by $6\varepsilon T$. All the other jobs are scheduled within the interval $[0, T)$. We potentially need to introduce one additional time-slot of the form $[t, t + 1)$ for each discarded job. We observe that for each discarded job j we can find a position to insert a time-slot for j since we assumed the precedence constraints to be transitive and we obtained a feasible schedule for all non-discarded jobs. Hence, the total length of the schedule is at most $(1 + 6\varepsilon)T$. ◀

Combining Lemma 10 and Lemma 1 gives us the following theorem.

► **Theorem 11.** *There exists an algorithm for the precedence constrained scheduling on identical parallel machines that is a $(1 + \varepsilon)$ -approximation and runs in time $n^{O_{m,\varepsilon}((\log n)^{m/\varepsilon})}$.*

References

- 1 Edward G Coffman and Ronald L Graham. Optimal scheduling for two-processor systems. *Acta informatica*, 1(3):200–213, 1972.
- 2 Devdatta Gangal and Abhiram Ranade. Precedence constrained scheduling in $(2-7/3p+1)$ optimal. *Journal of Computer and System Sciences*, 74(7):1139–1146, 2008. doi:10.1016/j.jcss.2008.04.001.
- 3 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.
- 4 Shashwat Garg. Quasi-PTAS for Scheduling with Precedences using LP Hierarchies. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 59:1–59:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.59.
- 5 Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, 45(9):1563–1581, 1966.
- 6 Shui Lam and Ravi Sethi. Worst case analysis of two scheduling algorithms. *SIAM Journal on Computing*, 6(3):518–536, 1977.
- 7 Elaine Levey and Thomas Rothvo. A $(1 + \epsilon)$ -approximation for makespan scheduling with precedence constraints using LP hierarchies. *SIAM Journal on Computing*, 50(3):STOC16–201, 2019.
- 8 Shi Li. Towards PTAS for precedence constrained scheduling via combinatorial algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2991–3010. SIAM, 2021.
- 9 Ola Svensson. Hardness of precedence constrained scheduling on identical machines. *SIAM Journal on Computing*, 40(5):1258–1274, 2011.

A Polynomial-Time Algorithm for 1/3-Approximate Nash Equilibria in Bimatrix Games

Argyrios Deligkas ✉

Royal Holloway, University of London, Egham, UK

Michail Fasoulakis ✉

Foundation for Research and Technology-Hellas (FORTH), Heraklion, Greece
Athens University of Economics and Business, Greece

Evangelos Markakis ✉

Athens University of Economics and Business, Greece

Abstract

Since the celebrated PPAD-completeness result for Nash equilibria in bimatrix games, a long line of research has focused on polynomial-time algorithms that compute ε -approximate Nash equilibria. Finding the best possible approximation guarantee that we can have in polynomial time has been a fundamental and non-trivial pursuit on settling the complexity of approximate equilibria. Despite a significant amount of effort, the algorithm of Tsaknakis and Spirakis [36], with an approximation guarantee of $(0.3393 + \delta)$, remains the state of the art over the last 15 years. In this paper, we propose a new refinement of the Tsaknakis-Spirakis algorithm, resulting in a polynomial-time algorithm that computes a $(\frac{1}{3} + \delta)$ -Nash equilibrium, for any constant $\delta > 0$. The main idea of our approach is to go beyond the use of convex combinations of primal and dual strategies, as defined in the optimization framework of [36], and enrich the pool of strategies from which we build the strategy profiles that we output in certain bottleneck cases of the algorithm.

2012 ACM Subject Classification Theory of computation → Exact and approximate computation of equilibria

Keywords and phrases bimatrix games, approximate Nash equilibria

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.41

Related Version *Full Version*: <https://arxiv.org/abs/2204.11525>

Funding *Evangelos Markakis*: This work has been 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 Number: HFRI-FM17-3512).

Acknowledgements The authors would like to thank Marcin Jurdziński for some initial discussions on the Tsaknakis-Spirakis algorithm.

1 Introduction

The notion of *Nash equilibrium* has been undoubtedly a fundamental solution concept in strategic games, ever since the seminal result of Nash [33], on the existence of equilibria for all finite games. Nash’s theorem however is only *existential*; it only shows that such an equilibrium always exists, but it does not provide an efficient algorithm to find one. In fact, many years after the work of Nash, in a series of breakthrough results, it was proven that computing a Nash equilibrium is PPAD-complete [16], even for *bimatrix* games [10], which provides strong evidence that computing an equilibrium is an intractable problem.

These negative results have naturally led to the study of *approximate* Nash equilibria. In an ε -approximate Nash equilibrium (ε -NE), no player can increase her payoff more than ε , by unilaterally changing her strategy. In contrast to exact Nash equilibria, the relaxation



© Argyrios Deligkas, Michail Fasoulakis, and Evangelos Markakis;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 41; pp. 41:1–41:14

Leibniz International Proceedings in Informatics



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

to ϵ -NE does admit subexponential algorithms. More precisely, the quasi polynomial-time approximation scheme (QPTAS) of [27] can find an ϵ -NE in time $n^{O(\log n/\epsilon^2)}$, for a game with n available pure strategies per player. One can then wonder whether the QPTAS could be improved to a PTAS or even a FPTAS. Unfortunately this does not seem to be the case, as the result of Chen, Deng, and Teng [10] already ruled out the existence of an FPTAS, unless $\text{PPAD}=\text{P}$. Some years later, in another breakthrough result, Rubinfeld [35] showed that, assuming the exponential-time hypothesis for PPAD, there exists a very small, yet unspecified, constant ϵ^* such that finding an ϵ -NE requires quasi polynomial time for every constant $\epsilon < \epsilon^*$. This would rule out a PTAS too.

Although it seems unlikely to have a polynomial time algorithm for any $\epsilon > 0$, it is still important to identify the best constant ϵ for which we can have an efficient algorithm. In fact, this has been one of the fundamental questions of algorithmic game theory, that is still unresolved. Soon after the initial PPAD-hardness results of [10, 16], there was a flourish of works along this direction. Kontogiannis, Panagopoulou, and Spirakis [23] derived a polynomial-time algorithm for $\epsilon = 3/4$; Daskalakis, Mehta, and Papadimitriou [17, 18] improved it to $\epsilon = 1/2$ and $\epsilon \approx 0.382$; Bosse, Byrka, and Markakis [7] achieved $\epsilon = 0.364$; and finally Tsaknakis and Spirakis [36] attained a bound of $\epsilon = 0.3393 + \delta$, for any constant $\delta > 0$. Ever since this last work however, the progress on this front has stalled, and the result of Tsaknakis and Spirakis (referred to as the TS algorithm from now on) remains the state of the art over the last 15 years. It is particularly puzzling that so far, it has remained an open problem to even improve the approximation to $1/3 + \delta$ (even though it has been conjectured that such an approximation should be feasible). To make things worse, in the very recent work of [12], it was shown that the TS algorithm and its analysis are tight.

In order to beat the 0.3393-guarantee of the TS algorithm, it is instructive to understand first its bottleneck cases. At a high level, we can think of the algorithm as consisting of two phases: the *Descent phase* and the *Strategy-construction phase*. In the Descent phase, it performs “gradient descent” on the maximum regret among the two players, i.e., the maximum additional gain that a player can have by a unilateral deviation to another strategy. This process terminates at an approximate “stationary” point, i.e., a strategy profile such that any local change does not decrease the value of the maximum regret. When we reach a δ -stationary point for some small constant δ , the Strategy-construction phase begins. This phase performs a case analysis, based on certain relevant parameters of the game, and tries to decide which strategy profile to output in each of the five cases that arise.

In doing so, the algorithm has at its disposal the δ -stationary profile, along with a “dual” strategy profile (produced by solving the dual of the linear program used in the Descent phase). A close inspection reveals that one of these two profiles suffices to guarantee a $(\frac{1}{3} + \delta)$ -NE in three out of the five cases. In the remaining two cases, the algorithm outputs a convex combination of the stationary and the dual strategies, and this is where the bottleneck occurs, causing the algorithm to output a $(0.3393 + \delta)$ -NE.

Our contribution

We improve upon the state of the art and provide a polynomial-time algorithm for computing a $(\frac{1}{3} + \delta)$ -NE in bimatrix games, for any constant $\delta > 0$. More specifically, we modify sufficiently the TS algorithm by designing an *improved* Strategy-construction phase to handle the problematic cases of TS. Our main insights in doing so are as follows:

- Apart from convex combinations between primal (stationary) and dual strategies, we also consider best response strategies to such convex combinations. Hence, we enrich the pool of strategies, out of which we choose the profile to output in each case. As a result,

in the cases where the δ -stationary point or the dual profile (or their combinations) do not have the desired guarantee, we have one of the players use a carefully chosen convex combination between our newly defined strategies and her dual strategy.

- We produce a more refined case analysis, that is based on the values of some new auxiliary parameters (e.g., the quantities v_r, t_r and $\hat{\mu}$, defined in Section 4). These parameters encode payoff differences or regrets of the players for using specific strategies, and they help us in two ways. First, they are used to obtain improved upper bounds on the maximum regret of the δ -stationary profile (Section 4.1). Secondly, their values greatly help us in decomposing our analysis into convenient subcases in order to establish the approximation guarantee.

Further related work

A different notion of approximation of NE is that of ε -well-supported NE (ε -WSNE). In an ε -WSNE every player is required to place positive probability only to actions that are within ε of being best responses. Hence, ε -WSNE are more constrained than ε -NE, where the players can place a positive probability on any strategy. After a series of papers on the topic [24, 21], the currently best approximation is for $\varepsilon = 0.6528$ due to [14].

Another line of research has focused on more structured classes of bimatrix games such as: constant-rank games, where the matrix defined by the sum of the two payoff matrices has constant rank [1, 22, 31]; win-lose games, where the payoff for every pure action is either 0 or 1 [11, 13, 28]; sparse games, where there are only “a few” outcomes that yield a non-zero payoff for each player [9], imitation games, where the payoff matrix for one of the players is the identity matrix [29, 30, 32]; random games, where the payoff entries are drawn from certain distributions [4, 34]; symmetric games, where the payoff matrix of one player is the transpose of the other [15, 25]. In most of these classes, it has been possible to obtain improved approximation guarantees and have a better understanding of how to construct approximate equilibria.

Concerning quasi-polynomial algorithms, in addition to the QPTAS of [27], three new QPTASs have been obtained, which contain the original result of [27] as a special case: [5] gave a refined, parameterized, approximation scheme; [3] gave a QPTAS that can be applied to multi-player games as well; [19] gave a more general approach for approximation schemes for the existential theory of the reals. More recently, more negative results for ε -NE were derived: [26] gave an unconditional lower bound, based on the sum of squares hierarchy; [6] proved PPAD-hardness in the smoothed analysis setting; [8, 20, 2] gave quasi-polynomial time lower bounds for constrained ε -NE, under the exponential time hypothesis.

2 Preliminaries

In what follows, let $[n] := \{1, 2, \dots, n\}$, and let Δ^n denote the $(n - 1)$ -dimensional simplex. We focus on $n \times n$ bimatrix games, where n denotes the number of available pure strategies per player. Such games are defined by a pair $(R, C) \in [0, 1]^{n \times n}$ of two matrices: R and C are the payoff matrices for the *row* player and the *column* player respectively. We follow the usual assumption in the relevant literature that the matrices are normalized, so that all entries are in $[0, 1]$. It is also assumed without loss of generality, that both players have the same number of pure strategies, since otherwise one can add dummy strategies to equalize the rows and columns. The semantics of the payoff matrices are that when the row player picks a row $i \in [n]$, and the column player picks a column $j \in [n]$, then they receive a payoff of R_{ij} and C_{ij} respectively.

41:4 A Polynomial-Time Algorithm for 1/3-NE in Bimatrix Games

A *mixed strategy* is a probability distribution over $[n]$. We use $\mathbf{x} \in \Delta^n$ to denote a mixed strategy for the row player and x_i to denote the probability the player assigns to the pure strategy i . For the column player, we use $\mathbf{y} \in \Delta^n$ and y_i , respectively. If \mathbf{x} and \mathbf{y} are mixed strategies for the row and the column player, then we call (\mathbf{x}, \mathbf{y}) a (*mixed*) *strategy profile*. It is often also convenient to represent pure strategies as vectors. Hence, we will use the vector e_i , which has 1 at index i and zero elsewhere, to denote the i -th pure strategy, in other words the distribution where a player assigns probability one to play the pure strategy i .

Given a strategy profile (\mathbf{x}, \mathbf{y}) , the expected payoff of the row player is $\mathcal{R}(\mathbf{x}, \mathbf{y}) := \mathbf{x}^T R \mathbf{y}$, and the expected payoff of the column player is $\mathcal{C}(\mathbf{x}, \mathbf{y}) := \mathbf{x}^T C \mathbf{y}$. Thus, for a pure strategy e_i , the term $\mathcal{R}(e_i, \mathbf{y}) := \sum_j R_{ij} y_j$, denotes the expected payoff of the row player, when she plays the pure strategy i against strategy \mathbf{y} of the column player. Similarly, $\mathcal{C}(\mathbf{x}, e_j)$ is the expected payoff of the column player when she plays the pure strategy j against \mathbf{x} . We say that a pure strategy is a *best-response strategy* for a player if it maximizes her expected payoff against a chosen strategy of her opponent. So, under a strategy profile (\mathbf{x}, \mathbf{y}) , the set of pure best responses for the row player is $B_r(\mathbf{y}) := \{i \in [n] : \mathcal{R}(e_i, \mathbf{y}) = \max_{i' \in [n]} \mathcal{R}(e_{i'}, \mathbf{y})\}$, and for the column player, it is $B_c(\mathbf{x}) := \{j \in [n] : \mathcal{C}(\mathbf{x}, e_j) = \max_{j' \in [n]} \mathcal{C}(\mathbf{x}, e_{j'})\}$.

The *regret* of the row player at a profile (\mathbf{x}, \mathbf{y}) , is $\text{reg}_r(\mathbf{x}, \mathbf{y}) = \max_i \mathcal{R}(e_i, \mathbf{y}) - \mathcal{R}(\mathbf{x}, \mathbf{y})$ and the regret of the column player is $\text{reg}_c(\mathbf{x}, \mathbf{y}) = \max_j \mathcal{C}(\mathbf{x}, e_j) - \mathcal{C}(\mathbf{x}, \mathbf{y})$. The strategy profile (\mathbf{x}, \mathbf{y}) is an ε -*Nash equilibrium*, or ε -NE, if the regret of both players is bounded by $\varepsilon \in [0, 1]$, formally $\max\{\text{reg}_r(\mathbf{x}, \mathbf{y}), \text{reg}_c(\mathbf{x}, \mathbf{y})\} \leq \varepsilon$. If $\varepsilon = 0$, then the strategy profile (\mathbf{x}, \mathbf{y}) is an exact Nash equilibrium.

3 The Tsaknakis-Spirakis algorithm

In this section we give a description of the algorithm by [36] and we highlight the bottleneck cases, where it fails to provide a $(\frac{1}{3} + \delta)$ -approximation. In order to have a self-contained exposition, we also present some of the lemmas that are used in the analysis of [36], which are needed for our work as well.

The core of the algorithm is to consider the function $g(\mathbf{x}, \mathbf{y}) = \max\{\text{reg}_r(\mathbf{x}, \mathbf{y}), \text{reg}_c(\mathbf{x}, \mathbf{y})\}$, i.e., the maximum regret among the two players. Clearly, if we arrive at a profile (\mathbf{x}, \mathbf{y}) such that $g(\mathbf{x}, \mathbf{y}) \leq \varepsilon$, then (\mathbf{x}, \mathbf{y}) is an ε -Nash equilibrium. At a high level, one can think of TS as consisting of two phases: the *Descent* phase, and the *Strategy-construction* phase.

Descent Phase. During this phase, TS performs “gradient descent” on the function $g(\mathbf{x}, \mathbf{y})$, until it reaches a “*stationary point*”, i.e., a strategy profile such that any local change does not decrease the value of g . More concretely, every iteration of the Descent phase performs a series of steps: given the current profile under consideration, it equalizes the regrets of the players, then it solves an appropriate linear program to identify a feasible direction, and finally depending on the solution of the LP, it either updates the strategy profile, or it decides that it has reached an approximate stationary point.

The first step runs the RegretEqualization procedure described below. This procedure is based on solving a single linear program to equalize the regrets of the two players, and most importantly, it guarantees that the maximum regret does not increase.

RegretEqualization($\mathbf{x}_0, \mathbf{y}_0$)

Input: A strategy profile $(\mathbf{x}_0, \mathbf{y}_0)$.

Output: A strategy profile (\mathbf{x}, \mathbf{y}) such that $\text{reg}_r(\mathbf{x}, \mathbf{y}) = \text{reg}_c(\mathbf{x}, \mathbf{y}) \leq g(\mathbf{x}_0, \mathbf{y}_0)$.

1. If $\text{reg}_r(\mathbf{x}_0, \mathbf{y}_0) \geq \text{reg}_c(\mathbf{x}_0, \mathbf{y}_0)$, keep \mathbf{y}_0 fixed and solve the following linear program:

Minimize $\text{reg}_r(\mathbf{x}, \mathbf{y}_0)$

Such that $\text{reg}_r(\mathbf{x}, \mathbf{y}_0) \geq \text{reg}_c(\mathbf{x}, \mathbf{y}_0)$ and $\mathbf{x} \in \Delta^n$.

Return $(\mathbf{x}, \mathbf{y}_0)$, where \mathbf{x} is the solution of the linear program.

2. If $\text{reg}_r(\mathbf{x}_0, \mathbf{y}_0) < \text{reg}_c(\mathbf{x}_0, \mathbf{y}_0)$, keep \mathbf{x}_0 fixed and solve the following linear program:

Minimize $\text{reg}_c(\mathbf{x}_0, \mathbf{y})$

Such that $\text{reg}_c(\mathbf{x}_0, \mathbf{y}) \geq \text{reg}_r(\mathbf{x}_0, \mathbf{y})$ and $\mathbf{y} \in \Delta^n$.

Return $(\mathbf{x}_0, \mathbf{y})$, where \mathbf{y} is the solution of the linear program.

Given the output (\mathbf{x}, \mathbf{y}) of RegretEqualization, the next step is to either find a feasible direction to follow so as to decrease the maximum regret, or to decide that (\mathbf{x}, \mathbf{y}) is an approximate stationary point. This is enforced by solving the following linear program.

Primal Linear Program: Primal(\mathbf{x}, \mathbf{y})

minimize γ

s.t. $\gamma \geq \mathcal{R}(e_i, \mathbf{y}') - \mathcal{R}(\mathbf{x}, \mathbf{y}') - \mathcal{R}(\mathbf{x}', \mathbf{y}) + \mathcal{R}(\mathbf{x}, \mathbf{y}), \quad \forall i \in B_r(\mathbf{y}),$

$\gamma \geq \mathcal{C}(\mathbf{x}', e_j) - \mathcal{C}(\mathbf{x}', \mathbf{y}) - \mathcal{C}(\mathbf{x}, \mathbf{y}') + \mathcal{C}(\mathbf{x}, \mathbf{y}), \quad \forall j \in B_c(\mathbf{x}),$

$\mathbf{x}' \in \Delta^n, \quad \mathbf{y}' \in \Delta^n.$

It is proved in [36] that the solution of Primal(\mathbf{x}, \mathbf{y}) guarantees one of the following:

1. it either identifies a strategy profile $(\mathbf{x}', \mathbf{y}')$ such that the maximum regret can be strictly decreased by a constant fraction, if we move from (\mathbf{x}, \mathbf{y}) towards $(\mathbf{x}', \mathbf{y}')$;
2. or it decides that (\mathbf{x}, \mathbf{y}) is a δ -stationary point¹, which is the termination criterion of the descent.

Putting everything together, the Descent phase of the TS algorithm is described below, starting from some arbitrary initial strategy profile, and its main properties are captured by the following lemma.

► **Lemma 1** ([36]). *For any constant $\delta > 0$, the Descent phase computes a δ -stationary point, in time polynomial in $1/\delta$ and in the size of the game.*

Descent Phase

Input: A strategy profile (\mathbf{x}, \mathbf{y}) , a small constant $\delta > 0$ ($\delta \ll 1/3$).

Output: A δ -stationary profile $(\mathbf{x}_s, \mathbf{y}_s)$ with equal regrets.

1. Equalize the regrets of the two players, i.e., set $(\mathbf{x}, \mathbf{y}) \leftarrow \text{RegretEqualization}(\mathbf{x}, \mathbf{y})$.

2. Solve Primal(\mathbf{x}, \mathbf{y}) and compute \mathbf{x}', \mathbf{y}' , and γ .

3. If $\gamma - g(\mathbf{x}, \mathbf{y}) \geq -\delta$, set $\mathbf{x}_s \leftarrow \mathbf{x}, \mathbf{y}_s \leftarrow \mathbf{y}$ and stop.

4. Else, set $\mathbf{x} \leftarrow (1 - \frac{\delta}{\delta+2}) \cdot \mathbf{x} + \frac{\delta}{\delta+2} \cdot \mathbf{x}'$, $\mathbf{y} \leftarrow (1 - \frac{\delta}{\delta+2}) \cdot \mathbf{y} + \frac{\delta}{\delta+2} \cdot \mathbf{y}'$ and go to Step 1.

¹ This means that the directional derivative of $g(\mathbf{x}, \mathbf{y})$ is at least $-\delta$. For the definition of directional derivative, see [36].

Strategy-construction Phase. In this phase, the algorithm utilizes the dual linear program of Primal(\mathbf{x}, \mathbf{y}), in order to identify some alternative candidate strategies for the players.

Dual Linear Program: Dual(\mathbf{x}, \mathbf{y})

$$\begin{aligned}
& \text{maximize} && P \cdot \mathcal{R}(\mathbf{x}, \mathbf{y}) + Q \cdot \mathcal{C}(\mathbf{x}, \mathbf{y}) + a + b \\
& \text{s.t.} && p_i \geq 0, \quad i \in B_r(\mathbf{y}), \\
& && q_j \geq 0, \quad j \in B_c(\mathbf{x}), \\
& && P = \sum_{i \in B_r(\mathbf{y})} p_i, \quad Q = \sum_{j \in B_c(\mathbf{x})} q_j, \\
& && P + Q = 1, \\
& && a \leq \sum_{i \in B_r(\mathbf{y})} -\mathcal{R}(e_k, \mathbf{y}) \cdot p_i + \sum_{j \in B_c(\mathbf{x})} [-\mathcal{C}(e_k, \mathbf{y}) + C_{kj}] \cdot q_j, \quad 1 \leq k \leq n \\
& && b \leq \sum_{j \in B_c(\mathbf{x})} -\mathcal{C}(\mathbf{x}, e_l) \cdot q_j + \sum_{i \in B_r(\mathbf{y})} [-\mathcal{R}(\mathbf{x}, e_l) + R_{il}] \cdot p_i, \quad 1 \leq l \leq n.
\end{aligned}$$

Given the δ -stationary profile $(\mathbf{x}_s, \mathbf{y}_s)$ from the Descent phase, the algorithm solves Dual($\mathbf{x}_s, \mathbf{y}_s$) and computes the following (from the optimal dual variables).

- The dual strategy \mathbf{w} for the row player, where $w_i = p_i / \sum_{j \in B_r(\mathbf{y}_s)} p_j$, for $i \in B_r(\mathbf{y}_s)$, and 0 elsewhere; note that by construction, \mathbf{w} is a best-response strategy against \mathbf{y}_s .
- The dual strategy \mathbf{z} for the column player, where $z_i = q_i / \sum_{j \in B_c(\mathbf{x}_s)} q_j$, for $i \in B_r(\mathbf{x}_s)$, and 0 elsewhere; by construction, \mathbf{z} is a best-response strategy against \mathbf{x}_s .
- The dual variables $P, Q \in [0, 1]$, that are useful for the approximation analysis.

In addition, we define the following two quantities λ and μ , that help in parameterizing the maximum regret bound. These quantities are equal to the payoff difference of a player between the dual and the primal strategies, when the other player uses her dual strategy:

$$\lambda = \mathcal{R}(\mathbf{w}, \mathbf{z}) - \mathcal{R}(\mathbf{x}_s, \mathbf{z}), \quad \mu = \mathcal{C}(\mathbf{w}, \mathbf{z}) - \mathcal{C}(\mathbf{w}, \mathbf{y}_s). \quad (1)$$

Fact. Obviously, $\lambda \leq 1$, and $\mu \leq 1$ and furthermore, $\mathcal{R}(\mathbf{w}, \mathbf{z}) \geq \lambda$, and $\mathcal{C}(\mathbf{w}, \mathbf{z}) \geq \mu$. The algorithm then constructs and outputs a strategy profile as follows.

Strategy-Construction Phase

Input: A δ -stationary strategy profile $(\mathbf{x}_s, \mathbf{y}_s)$ from the Descent phase, the dual strategies \mathbf{w}, \mathbf{z} , and the parameters λ, μ .

1. If $\min\{\lambda, \mu\} \leq \frac{1}{2}$, then return $(\mathbf{x}_s, \mathbf{y}_s)$.
2. If $\min\{\lambda, \mu\} \geq \frac{2}{3}$, then return (\mathbf{w}, \mathbf{z}) .
3. If $\min\{\lambda, \mu\} > \frac{1}{2}$ and $\max\{\lambda, \mu\} \leq \frac{2}{3}$, then return $(\mathbf{x}_s, \mathbf{y}_s)$.
4. Else if $\lambda \geq \mu$, then return the strategy profile with the minimum regret between $\left(\frac{1}{1+\lambda-\mu} \cdot \mathbf{w} + \frac{\lambda-\mu}{1+\lambda-\mu} \cdot \mathbf{x}_s, \mathbf{z}\right)$ and $(\mathbf{x}_s, \mathbf{y}_s)$.
5. Else if $\lambda < \mu$, then return the strategy profile with the minimum regret between $\left(\mathbf{w}, \frac{1}{1+\mu-\lambda} \cdot \mathbf{z} + \frac{\mu-\lambda}{1+\mu-\lambda} \cdot \mathbf{y}_s\right)$ and $(\mathbf{x}_s, \mathbf{y}_s)$.

► **Theorem 2** ([36]). *For any constant $\delta > 0$, the TS algorithm computes in polynomial time a $(0.3393 + \delta)$ -NE.*

► **Remark 3.** One could also check all the proposed profiles of this phase at every iteration of the Descent phase, as presented in [36], and stop if we have reached already the desired approximation. But this does not affect the worst-case running time, which occurs when the Descent phase terminates at a δ -stationary point.

We present below some important lemmas from [36] that are needed in our analysis too.

The first, and most important, lemma below shows how $\text{Primal}(\mathbf{x}_s, \mathbf{y}_s)$ and $\text{Dual}(\mathbf{x}_s, \mathbf{y}_s)$ can be used to bound the value of the maximum regret, $g(\mathbf{x}_s, \mathbf{y}_s)$.

► **Lemma 4** (implied by [36]). *Let $(\mathbf{x}_s, \mathbf{y}_s)$ be a δ -stationary point produced by the Descent Phase, for a constant $\delta > 0$. Let also \mathbf{w}, \mathbf{z} and P , be derived by an optimal solution to $\text{Dual}(\mathbf{x}_s, \mathbf{y}_s)$, as seen before. Then, for any strategy profile $(\mathbf{x}', \mathbf{y}')$, it holds that*

$$g(\mathbf{x}_s, \mathbf{y}_s) \leq P \cdot (\mathcal{R}(\mathbf{w}, \mathbf{y}') - \mathcal{R}(\mathbf{x}', \mathbf{y}_s) - \mathcal{R}(\mathbf{x}_s, \mathbf{y}') + \mathcal{R}(\mathbf{x}_s, \mathbf{y}_s)) + (1 - P) \cdot (\mathcal{C}(\mathbf{x}', \mathbf{z}) - \mathcal{C}(\mathbf{x}', \mathbf{y}_s) - \mathcal{C}(\mathbf{x}_s, \mathbf{y}') + \mathcal{C}(\mathbf{x}_s, \mathbf{y}_s)) + \delta.$$

Lemma 4 plays a crucial role as it allows us to bound $g(\mathbf{x}_s, \mathbf{y}_s)$ in terms of λ, μ and P , by making appropriate choices for \mathbf{x}' and \mathbf{y}' . This is used both in the following lemma and in Lemma 11 of Section 4.

► **Lemma 5** ([36]). *Let $(\mathbf{x}_s, \mathbf{y}_s)$ be a δ -stationary point produced by the Descent phase, for a constant $\delta > 0$, and let P be obtained by an optimal solution of $\text{Dual}(\mathbf{x}_s, \mathbf{y}_s)$. It holds that $g(\mathbf{x}_s, \mathbf{y}_s) \leq \min\{P \cdot \lambda, (1 - P) \cdot \mu\} + \delta \leq \frac{\lambda \cdot \mu}{\lambda + \mu} + \delta \leq \frac{\lambda + \mu}{4} + \delta$.*

One may worry that the bound $\frac{\lambda \cdot \mu}{\lambda + \mu}$ is not well-defined when $\lambda + \mu = 0$. However, as we explain below, this is not a concern.

► **Corollary 6.** *We can assume that both $\lambda > 0$ and $\mu > 0$, otherwise $(\mathbf{x}_s, \mathbf{y}_s)$ is a δ -Nash equilibrium.*

The definitions of λ and μ , along with Lemma 5 can immediately be used to prove that Cases 1-3 from the Strategy-construction phase return a $(\frac{1}{3} + \delta)$ -Nash equilibrium. Hence, the bottleneck of the TS algorithm comes from Cases 4 and 5. In fact, it was also recently shown in [12] that the analysis of these cases in [36] is tight, and therefore one needs to come up with a different construction in order to obtain an improvement.

► **Lemma 7** ([36]). *Cases 1-3 from the Strategy-construction phase return a $(\frac{1}{3} + \delta)$ -Nash equilibrium.*

Proof. We will consider every case independently.

- If $\min\{\lambda, \mu\} \leq \frac{1}{2}$, by Lemma 5 we have that $g(\mathbf{x}_s, \mathbf{y}_s) \leq \frac{\lambda \cdot \mu}{\lambda + \mu} + \delta \leq \frac{\min\{\lambda, \mu\}}{\min\{\lambda, \mu\} + 1} + \delta \leq \frac{1/2}{1/2 + 1} + \delta \leq \frac{1}{3} + \delta$. Here, the second inequality comes from the fact that $\frac{\lambda \cdot \mu}{\lambda + \mu}$ is an increasing function of $\max\{\lambda, \mu\}$, and also $\max\{\lambda, \mu\} \leq 1$.
- If $\min\{\lambda, \mu\} \geq \frac{2}{3}$, then $g(\mathbf{w}, \mathbf{z}) \leq \max\{1 - \mathcal{R}(\mathbf{w}, \mathbf{z}), 1 - \mathcal{C}(\mathbf{w}, \mathbf{z})\}$. But since $\mathcal{R}(\mathbf{w}, \mathbf{z}) \geq \lambda$ and $\mathcal{C}(\mathbf{w}, \mathbf{z}) \geq \mu$, we have that the regret is at most $1 - \min\{\lambda, \mu\} \leq \frac{1}{3}$.
- If $\min\{\lambda, \mu\} > \frac{1}{2}$ and $\max\{\lambda, \mu\} \leq \frac{2}{3}$, by Lemma 5 we have $g(\mathbf{x}_s, \mathbf{y}_s) \leq \frac{\lambda \cdot \mu}{\lambda + \mu} + \delta \leq \frac{\frac{2}{3} \cdot \frac{2}{3}}{\frac{2}{3} + \frac{2}{3}} + \delta \leq \frac{1}{3} + \delta$, since $\frac{\lambda \cdot \mu}{\lambda + \mu}$ is an increasing function of λ and μ . ◀

Thus, in the next section, we will focus on the remaining cases, when $\min\{\lambda, \mu\} \in (\frac{1}{2}, \frac{2}{3}]$ and $\max\{\lambda, \mu\} \in (\frac{2}{3}, 1]$.

4 Improved Strategy-construction Phase

In this section we replace Cases 4 and 5 from the original TS algorithm in order to bypass the bottleneck in the approximation. To do so, we utilize the δ -stationary point $(\mathbf{x}_s, \mathbf{y}_s)$, the dual strategies \mathbf{w}, \mathbf{z} , their convex combinations and best-response strategies to such

combinations. We then perform a more refined analysis and prove that in every case we can efficiently construct a tailored strategy profile that is a $(\frac{1}{3} + \delta)$ -Nash equilibrium. We note that all missing proofs from this section can be found in the full version of our work.

Our new Strategy-construction phase works as follows.

Improved Strategy-construction Phase

Input: A δ -stationary point $(\mathbf{x}_s, \mathbf{y}_s)$ from the Descent phase, the dual strategies \mathbf{w} , \mathbf{z} , and the parameters λ, μ .

1. If $\min\{\lambda, \mu\} \leq \frac{1}{2}$, then return $(\mathbf{x}_s, \mathbf{y}_s)$.
2. If $\min\{\lambda, \mu\} \geq \frac{2}{3}$, then return (\mathbf{w}, \mathbf{z}) .
3. If $\min\{\lambda, \mu\} > \frac{1}{2}$ and $\max\{\lambda, \mu\} \leq \frac{2}{3}$, then return $(\mathbf{x}_s, \mathbf{y}_s)$.
4. If $\frac{1}{2} < \lambda \leq \frac{2}{3} < \mu$:
 - Set $\hat{\mathbf{y}} = \frac{1}{2} \cdot \mathbf{y}_s + \frac{1}{2} \cdot \mathbf{z}$.
 - Find a best response $\hat{\mathbf{w}}$ against $\hat{\mathbf{y}}$.
 - Set $t_r = \mathcal{R}(\hat{\mathbf{w}}, \hat{\mathbf{y}}) - \mathcal{R}(\mathbf{w}, \hat{\mathbf{y}})$; $v_r = \mathcal{R}(\mathbf{w}, \mathbf{y}_s) - \mathcal{R}(\hat{\mathbf{w}}, \mathbf{y}_s)$; $\hat{\mu} = \mathcal{C}(\hat{\mathbf{w}}, \mathbf{z}) - \mathcal{C}(\hat{\mathbf{w}}, \mathbf{y}_s)$.
 - 4.1 If $v_r + t_r \geq \frac{\mu - \lambda}{2}$ and $\hat{\mu} \geq \mu - v_r - t_r$, then set $p = \frac{2 \cdot (v_r + t_r) - (\mu - \lambda)}{2 \cdot (v_r + t_r)}$ and return the strategy profile with the minimum regret among $(p \cdot \mathbf{w} + (1 - p) \cdot \hat{\mathbf{w}}, \mathbf{z})$ and $(\mathbf{x}_s, \mathbf{y}_s)$.
 - 4.2 Else, set $q = \frac{1 - \mu/2 - t_r}{1 + \mu/2 - \lambda - t_r}$ and return the strategy profile with the minimum regret among $(\mathbf{w}, (1 - q) \cdot \hat{\mathbf{y}} + q \cdot \mathbf{z})$ and $(\mathbf{x}_s, \mathbf{y}_s)$.
5. If $\frac{1}{2} < \mu \leq \frac{2}{3} < \lambda$ (symmetric to Case 4):
 - Set $\hat{\mathbf{x}} = \frac{1}{2} \cdot \mathbf{x}_s + \frac{1}{2} \cdot \mathbf{w}$.
 - Find a best response $\hat{\mathbf{z}}$ against $\hat{\mathbf{x}}$.
 - Set $t_c = \mathcal{C}(\hat{\mathbf{x}}, \hat{\mathbf{z}}) - \mathcal{C}(\hat{\mathbf{x}}, \mathbf{z})$; $v_c = \mathcal{C}(\mathbf{x}_s, \mathbf{z}) - \mathcal{C}(\mathbf{x}_s, \hat{\mathbf{z}})$; $\hat{\lambda} = \mathcal{R}(\mathbf{w}, \hat{\mathbf{z}}) - \mathcal{R}(\mathbf{x}_s, \hat{\mathbf{z}})$.
 - 5.1 If $v_c + t_c \geq \frac{\lambda - \mu}{2}$ and $\hat{\lambda} \geq \lambda - v_c - t_c$, then set $p = \frac{2 \cdot (v_c + t_c) - (\lambda - \mu)}{2 \cdot (v_c + t_c)}$ and return the strategy profile with the minimum regret among $(\mathbf{w}, p \cdot \mathbf{z} + (1 - p) \cdot \hat{\mathbf{z}})$ and $(\mathbf{x}_s, \mathbf{y}_s)$.
 - 5.2 Else, set $q = \frac{1 - \lambda/2 - t_c}{1 + \lambda/2 - \mu - t_c}$ and return the strategy profile with the minimum regret among $((1 - q) \cdot \hat{\mathbf{x}} + q \cdot \mathbf{w}, \mathbf{z})$ and $(\mathbf{x}_s, \mathbf{y}_s)$.

Note that Cases 1-3 are identical to the Strategy-construction phase of the TS algorithm. Thus, by Lemma 7 they return a $(\frac{1}{3} + \delta)$ -Nash equilibrium. The new part concerns Cases 4 and 5. The analysis in both cases is based on certain auxiliary parameters (v_r, t_r and $\hat{\mu}$ for Case 4 and analogously for Case 5), that we define in the statement of the algorithm. These parameters encode payoff differences or regrets of the players for using specific strategies, and they help us decompose the problem into convenient subcases, so as to obtain better upper bounds on the maximum regret.

Our main result is as follows:

► **Theorem 8.** *For any constant $\delta > 0$, we can compute in polynomial-time a $(\frac{1}{3} + \delta)$ -Nash equilibrium.*

To prove the theorem, it suffices to analyze Case 4, where $\frac{1}{2} < \lambda \leq \frac{2}{3} < \mu$, since Case 5 is symmetric to Case 4 and is analyzed in exactly the same way.

Intuition and Roadmap. The overall analysis in the sequel looks rather technical, therefore, we will first provide some elaboration on the choices that the algorithm makes in Case 4. The first crucial component in the design of the new algorithm is that the upper bounds on the regret of the δ -stationary point $(\mathbf{x}_s, \mathbf{y}_s)$, obtained in Lemma 5, can be further refined based on the values of the parameters $\lambda, \mu, \hat{\mu}, v_r$. This is precisely implemented in Section 4.1 with Lemmas 11, 12, and 13. Once this is done, we then try to answer the following question: Whenever $(\mathbf{x}_s, \mathbf{y}_s)$ does not provide a $(\frac{1}{3} + \delta)$ -approximation, which profiles can form alternative candidates for a better performance? One idea is to exploit the dual strategies \mathbf{w} , and \mathbf{z} , as was also done in [36]. However, the profile (\mathbf{w}, \mathbf{z}) may not be a $(\frac{1}{3} + \delta)$ -equilibrium either (in most cases). A next attempt then is to consider appropriate convex combinations of the primal and the dual strategy for each player, i.e., a combination of \mathbf{x}_s and \mathbf{w} for the row player and \mathbf{y}_s and \mathbf{z} for the column player. Unfortunately, this again does not work in all cases. But one next step is to also take into consideration best-response strategies against such convex combinations. E.g., the strategy $\hat{\mathbf{w}}$ defined in Case 4 is a best response to the equiprobable combination of \mathbf{y}_s and \mathbf{z} . This completes our weaponry, and at the end, in all subcases of Case 4, we consider profiles where the row player uses a convex combination of \mathbf{w} and $\hat{\mathbf{w}}$, and the column player selects a combination between her primal and dual strategies, \mathbf{y}_s and \mathbf{z} . Analogous profiles with the roles of the players reversed are constructed for Case 5 too. Finally, we also know that whenever $(\mathbf{x}_s, \mathbf{y}_s)$ does not attain a $(\frac{1}{3} + \delta)$ -approximation, this restricts the relation between the parameters $\lambda, \mu, \hat{\mu}$ and v_r due to the lemmas of Section 4.1. This is exploitable for us in the sense that it allows us to construct the exact coefficients for the convex combinations that we use so as to have the desired approximation.

To proceed, we start with two helpful observations, which are used repeatedly for the analysis of Cases 4.1 and 4.2.

► **Lemma 9.** *It holds that $\mathcal{R}(\hat{\mathbf{w}}, \mathbf{z}) \geq \lambda + v_r + 2t_r$.*

Proof. By the definition of t_r , inside Case 4, we have that it holds that $\mathcal{R}(\hat{\mathbf{w}}, \hat{\mathbf{y}}) = \mathcal{R}(\mathbf{w}, \hat{\mathbf{y}}) + t_r$. Hence,

$$\frac{\mathcal{R}(\hat{\mathbf{w}}, \mathbf{y}_s)}{2} + \frac{\mathcal{R}(\hat{\mathbf{w}}, \mathbf{z})}{2} = \mathcal{R}(\hat{\mathbf{w}}, \hat{\mathbf{y}}) = \frac{\mathcal{R}(\mathbf{w}, \mathbf{y}_s)}{2} + \frac{\mathcal{R}(\mathbf{w}, \mathbf{z})}{2} + t_r \Rightarrow$$

$$\mathcal{R}(\hat{\mathbf{w}}, \mathbf{z}) = (\mathcal{R}(\mathbf{w}, \mathbf{y}_s) - \mathcal{R}(\hat{\mathbf{w}}, \mathbf{y}_s)) + \mathcal{R}(\mathbf{w}, \mathbf{z}) + 2t_r \geq \lambda + v_r + 2t_r,$$

since $\mathcal{R}(\mathbf{w}, \mathbf{y}_s) - \mathcal{R}(\hat{\mathbf{w}}, \mathbf{y}_s) = v_r$, and $\mathcal{R}(\mathbf{w}, \mathbf{z}) \geq \lambda$ (by the fact after Equation (1)). ◀

► **Corollary 10.** *It holds that $v_r \leq 1 - \lambda - 2t_r$, or equivalently $t_r \leq \frac{1 - \lambda - v_r}{2}$.*

Proof. By the previous lemma we have $\mathcal{R}(\hat{\mathbf{w}}, \mathbf{z}) \geq \lambda + v_r + 2t_r \Rightarrow v_r \leq 1 - \lambda - 2t_r$, since $\mathcal{R}(\hat{\mathbf{w}}, \mathbf{z}) \leq 1$. ◀

4.1 Bounding the regret of δ -stationary points

In this subsection, we establish three important lemmas that provide different ways of bounding the maximum regret of any δ -stationary point. The first of these lemmas is an improvement over [36], where we add a third upper bound for the δ -stationary point, in addition to the bounds stated in Lemma 5 from Section 3.

► **Lemma 11.** *Let $(\mathbf{x}_s, \mathbf{y}_s)$ be a δ -stationary point with $\delta \geq 0$, and let P be obtained by an optimal solution of $\text{Dual}(\mathbf{x}_s, \mathbf{y}_s)$, as the sum of the dual variables: $P = \sum_{i \in B_r(\mathbf{y}_s)} p_i$. It holds that $g(\mathbf{x}_s, \mathbf{y}_s) \leq \min\{P \cdot \lambda, (1 - P) \cdot \mu, P \cdot v_r + (1 - P) \cdot \hat{\mu}\} + \delta$.*

41:10 A Polynomial-Time Algorithm for 1/3-NE in Bimatrix Games

Proof. By Lemma 5 it holds that $g(\mathbf{x}_s, \mathbf{y}_s) \leq \min\{P \cdot \lambda, (1-P) \cdot \mu\} + \delta$. So, it suffices to prove that $g(\mathbf{x}_s, \mathbf{y}_s) \leq P \cdot v_r + (1-P) \cdot \hat{\mu} + \delta$. This follows from Lemma 4 when we set $(\mathbf{x}', \mathbf{y}') = (\hat{\mathbf{w}}, \mathbf{y}_s)$. Indeed, in this case we have $g(\mathbf{x}_s, \mathbf{y}_s) \leq P \cdot (\mathcal{R}(\mathbf{w}, \mathbf{y}_s) - \mathcal{R}(\hat{\mathbf{w}}, \mathbf{y}_s) - \mathcal{R}(\mathbf{x}_s, \mathbf{y}_s) + \mathcal{R}(\mathbf{x}_s, \mathbf{y}_s)) + (1-P) \cdot (\mathcal{C}(\hat{\mathbf{w}}, \mathbf{z}) - \mathcal{C}(\hat{\mathbf{w}}, \mathbf{y}_s) - \mathcal{C}(\mathbf{x}_s, \mathbf{y}_s) + \mathcal{C}(\mathbf{x}_s, \mathbf{y}_s)) = P \cdot v_r + (1-P) \cdot \hat{\mu} + \delta$, by the definitions of v_r and $\hat{\mu}$. \blacktriangleleft

The remaining two lemmas help in attaining a more fine-grained analysis on upper bounding the regret of the players, under the restrictions on the values of λ and μ in Case 4.

► **Lemma 12.** *Let $(\mathbf{x}_s, \mathbf{y}_s)$ be a δ -stationary point with $\delta \geq 0$, and let $\hat{\mu} \geq v_r$, and $\lambda > \frac{1}{2}$. Then, it holds that $g(\mathbf{x}_s, \mathbf{y}_s) \leq \frac{\hat{\mu} \cdot \lambda}{\lambda + \hat{\mu} - v_r} + \delta$.*

Proof. By Lemma 11 we have $g(\mathbf{x}_s, \mathbf{y}_s) \leq \min\{P \cdot \lambda, P \cdot v_r + (1-P) \cdot \hat{\mu}\} + \delta$. Note that $P \cdot \lambda$ is an increasing linear function of P , and $P \cdot v_r + (1-P) \cdot \hat{\mu}$ is a decreasing linear function of P , because $\hat{\mu} \geq v_r$. Therefore, the maximum of the minimum of these two functions is achieved at the point where they are equal, which is for $P' = \frac{\hat{\mu}}{\lambda + \hat{\mu} - v_r}$, as long as $P' \in [0, 1]$ (recall that P is constrained to belong to this interval). To check that P' is a valid point, observe first that since $\lambda > \frac{1}{2}$ and $\hat{\mu} \geq v_r$, the denominator of P' is positive. Also, again using that $\lambda > \frac{1}{2}$, Corollary 10 implies that $v_r \leq 1 - \lambda \leq \frac{1}{2}$, hence $\lambda > v_r$, which means that $P' \in [0, 1]$. Thus, $g(\mathbf{x}_s, \mathbf{y}_s) \leq \lambda \cdot P' + \delta = \frac{\hat{\mu} \cdot \lambda}{\lambda + \hat{\mu} - v_r} + \delta$. \blacktriangleleft

► **Lemma 13.** *Let $(\mathbf{x}_s, \mathbf{y}_s)$ be a δ -stationary point with $\delta \geq 0$, and let $\hat{\mu} < v_r$, $\lambda > \frac{1}{2}$, and $\mu > \frac{2}{3}$. Then, it holds that $g(\mathbf{x}_s, \mathbf{y}_s) \leq \frac{v_r \cdot \mu}{\mu - \hat{\mu} + v_r} + \delta$.*

4.2 Case 4.1 of the Improved Strategy-construction Phase

We now analyze the approximation we obtain, when we fall into Case 4.1 of the algorithm. We establish that either the δ -stationary point has the desired approximation or otherwise, this is achieved by having the row player use an appropriate convex combination of \mathbf{w} and $\hat{\mathbf{w}}$ and the column player play the dual strategy \mathbf{z} .

► **Lemma 14.** *If $v_r + t_r \geq \frac{\mu - \lambda}{2}$, and $\hat{\mu} \geq \mu - v_r - t_r$, then for the strategy profile $(p \cdot \mathbf{w} + (1-p) \cdot \hat{\mathbf{w}}, \mathbf{z})$, with $p = \frac{2 \cdot (v_r + t_r) - (\mu - \lambda)}{2 \cdot (v_r + t_r)}$, the payoff of both the row and the column player is at least $\frac{\lambda + \mu}{2}$.*

Proof. Note first that under the assumptions of the lemma, and since $\mu > \lambda$, the parameter p is a valid probability. For the row player, we have that her payoff is

$$\begin{aligned} \mathcal{R}(p \cdot \mathbf{w} + (1-p) \cdot \hat{\mathbf{w}}, \mathbf{z}) &= p \cdot \mathcal{R}(\mathbf{w}, \mathbf{z}) + (1-p) \cdot \mathcal{R}(\hat{\mathbf{w}}, \mathbf{z}) \\ &\geq p \cdot \lambda + (1-p) \cdot \lambda + (1-p) \cdot (v_r + t_r) \quad (\text{from Lemma 9}) \\ &= \lambda + (1-p) \cdot (v_r + t_r) \\ &= \lambda + \frac{(\mu - \lambda)}{2} \quad \left(\text{since } 1-p = \frac{\mu - \lambda}{2 \cdot (v_r + t_r)} \right) \\ &= \frac{\lambda + \mu}{2}. \end{aligned}$$

For the column player we have that her payoff is

$$\begin{aligned}
\mathcal{C}(p \cdot \mathbf{w} + (1-p) \cdot \hat{\mathbf{w}}, \mathbf{z}) &= p \cdot \mathcal{C}(\mathbf{w}, \mathbf{z}) + (1-p) \cdot \mathcal{C}(\hat{\mathbf{w}}, \mathbf{z}) \\
&\geq p \cdot \mu + (1-p) \cdot \hat{\mu} \quad (\text{Since } \hat{\mu} = \mathcal{C}(\hat{\mathbf{w}}, \mathbf{z}) - \mathcal{C}(\hat{\mathbf{w}}, \mathbf{y}_s) \leq \mathcal{C}(\hat{\mathbf{w}}, \mathbf{z})) \\
&\geq p \cdot \mu + (1-p) \cdot \mu - (1-p) \cdot (v_r + t_r) \quad (\text{Since } \hat{\mu} \geq \mu - v_r - t_r) \\
&= \mu - \frac{(\mu - \lambda)}{2} \quad \left(\text{since } 1-p = \frac{\mu - \lambda}{2 \cdot (v_r + t_r)} \right) \\
&= \frac{\mu + \lambda}{2}. \quad \blacktriangleleft
\end{aligned}$$

► **Lemma 15.** *Let $p \in [0, 1]$, be such that $\mathcal{R}(p \cdot \mathbf{w} + (1-p) \cdot \hat{\mathbf{w}}, \mathbf{z}) \geq \frac{\lambda + \mu}{2}$, and $\mathcal{C}(p \cdot \mathbf{w} + (1-p) \cdot \hat{\mathbf{w}}, \mathbf{z}) \geq \frac{\lambda + \mu}{2}$. Then, either $(\mathbf{x}_s, \mathbf{y}_s)$ is a $(\frac{1}{3} + \delta)$ -Nash equilibrium, or $(p \cdot \mathbf{w} + (1-p) \cdot \hat{\mathbf{w}}, \mathbf{z})$ is a $\frac{1}{3}$ -Nash equilibrium.*

Proof. The regret of either player at the strategy profile $(p \cdot \mathbf{w} + (1-p) \cdot \hat{\mathbf{w}}, \mathbf{z})$ is at most $1 - \frac{\lambda + \mu}{2}$, since the payoff of any player is no less than $\frac{\lambda + \mu}{2}$ and the best-response payoff is at most 1. On the other hand, by Lemma 5 the regret of each player at the δ -stationary point $(\mathbf{x}_s, \mathbf{y}_s)$ is at most $\frac{\lambda + \mu}{4} + \delta$. Thus, if $\lambda + \mu \leq \frac{4}{3}$, then $g(\mathbf{x}_s, \mathbf{y}_s) \leq \frac{1}{3} + \delta$. Otherwise, the maximum regret at the profile $(p \cdot \mathbf{w} + (1-p) \cdot \hat{\mathbf{w}}, \mathbf{z})$ is at most $1 - \frac{\lambda + \mu}{2} \leq \frac{1}{3}$. ◀

4.3 Case 4.2 of the Strategy-construction phase

In this case it holds that either $v_r + t_r < \frac{\mu - \lambda}{2}$ or $\hat{\mu} < \mu - v_r - t_r$. It turns out that this is a technically more intriguing case, and the reason is that the parameters are less constrained, compared to Case 4.1. As a result, we need to consider different subcases in order to have tighter upper bounds. We recall that the algorithm in this case outputs either $(\mathbf{x}_s, \mathbf{y}_s)$, or a profile where the row player selects her dual strategy \mathbf{w} , which is a best response against \mathbf{y}_s , and the column player plays a convex combination between $\hat{\mathbf{y}}$ and \mathbf{z} , which by the definition of $\hat{\mathbf{y}}$, is a convex combination of her primal strategy \mathbf{y}_s and her dual strategy \mathbf{z} .

► **Lemma 16.** *The regret of the row player at $(\mathbf{w}, \hat{\mathbf{y}})$ is t_r , and the regret of the column player is at most $1 - \frac{\mu}{2}$.*

Proof. By definition, the regret of the row player is t_r , since $\hat{\mathbf{w}}$ is a best-response strategy against $\hat{\mathbf{y}}$. On the other hand, recall by the definition of μ , that $\mathcal{C}(\mathbf{w}, \mathbf{z}) \geq \mu$. So, we have that $\mathcal{C}(\mathbf{w}, \hat{\mathbf{y}}) = \frac{\mathcal{C}(\mathbf{w}, \mathbf{y}_s)}{2} + \frac{\mathcal{C}(\mathbf{w}, \mathbf{z})}{2} \geq \frac{\mathcal{C}(\mathbf{w}, \mathbf{z})}{2} \geq \frac{\mu}{2}$. Thus, since the maximum payoff is less than or equal to 1, we have that the regret of the column player is at most $1 - \frac{\mu}{2}$. ◀

We now quantify the regret of the players at the profile $(\mathbf{w}, (1-q) \cdot \hat{\mathbf{y}} + q \cdot \mathbf{z})$, that is considered by the algorithm. In particular, we obtain an upper bound as a function of the parameters λ, μ , and t_r .

► **Lemma 17.** *Consider the strategy profile $(\mathbf{w}, (1-q) \cdot \hat{\mathbf{y}} + q \cdot \mathbf{z})$ with $q = \frac{1 - \mu/2 - t_r}{1 + \mu/2 - \lambda - t_r}$. Then, the regret of each player is no greater than $q \cdot (1 - \lambda) + (1 - q) \cdot t_r = \frac{1 - \mu/2 - t_r - \lambda + \mu \cdot \lambda/2 + \mu \cdot t_r}{1 + \mu/2 - \lambda - t_r}$.*

We now come to the core of the proof and establish that either the δ -stationary point, or the strategy profile $(\mathbf{w}, (1-q) \cdot \hat{\mathbf{y}} + q \cdot \mathbf{z})$ yields a good approximation. This is established by the following lemma.

► **Lemma 18.** *Under the assumptions of Case 4.2, either $(\mathbf{x}_s, \mathbf{y}_s)$ is a $(\frac{1}{3} + \delta)$ -Nash equilibrium, or $(\mathbf{w}, (1-q) \cdot \hat{\mathbf{y}} + q \cdot \mathbf{z})$ with $q = \frac{1 - \mu/2 - t_r}{1 + \mu/2 - \lambda - t_r}$, is a $\frac{1}{3}$ -Nash equilibrium.*

41:12 A Polynomial-Time Algorithm for 1/3-NE in Bimatrix Games

Proof. Since we are in Case 4.2, where either $v_r + t_r < \frac{\mu - \lambda}{2}$, or $\hat{\mu} < \mu - v_r - t_r$, we will split the analysis into further subcases, so that we have a more concrete relation between the relevant parameters in each subcase. More precisely, we will consider the following three subcases.

4.2(i) $v_r + t_r < \frac{\mu - \lambda}{2}$.

4.2(ii) $\hat{\mu} < \mu - v_r - t_r$, and $\hat{\mu} \geq v_r$.

4.2(iii) $\hat{\mu} < \mu - v_r - t_r$, and $\hat{\mu} < v_r$.

So far, we have not been able to have a unifying argument for all these different subcases. Consequently, we need to proceed with a separate analysis for each of them. Due to space constraints, we defer the first two subcases to the full version of this work, and we present here only Subcase 4.2(iii). We note also that among them, it was Subcase 4.2(ii) that turned out to be the lengthier and technically most subtle in obtaining the approximation bound.

Subcase 4.2(iii). Let us begin by observing that if $v_r \leq \frac{1}{3}$, then the strategy profile $(\mathbf{x}_s, \mathbf{y}_s)$ is a $(\frac{1}{3} + \delta)$ -NE. Indeed, from Lemma 13, and since $\hat{\mu} < v_r$, we get that the approximation guarantee of the stationary strategy profile is $\frac{v_r \cdot \mu}{\mu - \hat{\mu} + v_r} + \delta \leq v_r + \delta \leq \frac{1}{3} + \delta$. Hence, in what follows, we will assume that $v_r > \frac{1}{3}$. So, from Corollary 10 we have that

$$t_r \leq \frac{1 - \lambda - v_r}{2} < \frac{\frac{2}{3} - \lambda}{2} = \frac{1}{3} - \frac{\lambda}{2}. \quad (2)$$

Assume now for the sake of contradiction, that $(\mathbf{w}, (1 - q) \cdot \hat{\mathbf{y}} + q \cdot \mathbf{z})$ is not a $\frac{1}{3}$ -NE, i.e., the maximum regret is higher than $1/3$. Combining this with the bound of Lemma 17 on the regret, and by simplifying the resulting expression, we get the following inequality:

$$\frac{\lambda\mu}{2} - \frac{2\mu}{3} - \frac{2\lambda}{3} + \left(\mu - \frac{2}{3}\right) \cdot t_r + \frac{2}{3} > 0. \quad (3)$$

We will focus on the LHS of (3) and we will upper bound it by a non-positive value. Suppose that the regret at the stationary point $(\mathbf{x}_s, \mathbf{y}_s)$ is greater than $(\frac{1}{3} + \delta)$. This means from Lemma 5 that $\frac{\lambda \cdot \mu}{\lambda + \mu} > \frac{1}{3}$, which implies that $\mu > \frac{\lambda}{3\lambda - 1}$. In addition, observe that since $\mu > 2/3$, the LHS of (3) is increasing with t_r . Hence, if we use Inequality (2), we get that the LHS of (3) is upper bounded by

$$\begin{aligned} \frac{\lambda\mu}{2} - \frac{2\mu}{3} - \frac{2\lambda}{3} + \left(\mu - \frac{2}{3}\right) \cdot \left(\frac{1}{3} - \frac{\lambda}{2}\right) + \frac{2}{3} &= -\frac{\mu}{3} - \frac{\lambda}{3} + \frac{4}{9} \\ &< -\frac{\lambda}{9\lambda - 3} - \frac{\lambda}{3} + \frac{4}{9} \quad \left(\text{since } \mu > \frac{\lambda}{3\lambda - 1}\right). \end{aligned}$$

It is easy to verify though that this quantity is non-positive for every $\lambda \in [\frac{1}{2}, \frac{2}{3}]$, which in turn contradicts Inequality (3). \blacktriangleleft

5 Discussion

Our algorithm is the first improvement for a foundational problem after 15 years, during which progress had stalled. We hope that our result will again ignite the spark for actively studying ε -NE in bimatrix games. There is still a large gap between the quasi polynomial-time lower bound for some very small constant ε^* from [35], and our newly-established upper bound of $1/3$. We conjecture that closing this gap requires radically new ideas.

Our result has some extra positive consequences for games with more than two players. In [7] it was shown that if we have an algorithm that finds an α -Nash equilibrium in a $(k - 1)$ -player game, then in polynomial time we can compute a $(\frac{1}{2-\alpha})$ -NE for any k -player game. Thus, our algorithm improves the state of the art for k -player normal-form games, for any $k > 2$. Namely, we get a $(0.6 + \delta)$ -NE for three-player games, a $(5/7 + \delta)$ -NE for four-player games, and so on.

References

- 1 Bharat Adsul, Jugal Garg, Ruta Mehta, and Milind Sohoni. Rank-1 bimatrix games: a homeomorphism and a polynomial time algorithm. In *Proceedings of STOC*, pages 195–204, 2011.
- 2 Per Austrin, Mark Braverman, and Eden Chlamtáč. Inapproximability of np-complete variants of Nash equilibrium. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 13–25. Springer, 2011.
- 3 Yakov Babichenko, Siddharth Barman, and Ron Peretz. Empirical distribution of equilibrium play and its testing application. *Math. Oper. Res.*, 42(1):15–29, 2017.
- 4 Imre Bárány, Santosh Vempala, and Adrian Vetta. Nash equilibria in random games. *Random Structures & Algorithms*, 31(4):391–405, 2007.
- 5 Siddharth Barman. Approximating Nash equilibria and dense subgraphs via an approximate version of Carathéodory’s theorem. *SIAM J. Comput.*, 47(3):960–981, 2018.
- 6 Shant Boodaghians, Joshua Brakensiek, Samuel B. Hopkins, and Aviad Rubinfeld. Smoothed complexity of 2-player Nash equilibria. In *Proceedings of FOCS*, pages 271–282, 2020.
- 7 Hartwig Bosse, Jaroslaw Byrka, and Evangelos Markakis. New algorithms for approximate Nash equilibria in bimatrix games. *Theoretical Computer Science*, 411(1):164–173, 2010.
- 8 Mark Braverman, Young Kun-Ko, and Omri Weinstein. Approximating the best Nash equilibrium in $n^{o(\log n)}$ -time breaks the exponential time hypothesis. In *Proceedings of SODA*, pages 970–982. SIAM, 2015.
- 9 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Sparse games are hard. In *Proceedings of WINE*, pages 262–273, 2006.
- 10 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3), 2009.
- 11 Xi Chen, Shang-Hua Teng, and Paul Valiant. The approximation complexity of win-lose games. In *Proceedings of SODA*, volume 7, pages 159–168, 2007.
- 12 Zhaohua Chen, Xiaotie Deng, Wenhan Huang, Hanyu Li, and Yuhao Li. On tightness of the Tsaknakis-Spirakis algorithm for approximate Nash equilibrium. In *Proceedings of SAGT*, volume 12885, pages 97–111, 2021.
- 13 Bruno Codenotti and Daniel Štefankovič. On the computational complexity of Nash equilibria for $(0,1)$ bimatrix games. *Information Processing Letters*, 94(3):145–150, 2005.
- 14 Artur Czumaj, Argyrios Deligkas, Michail Fasoulakis, John Fearnley, Marcin Jurdziński, and Rahul Savani. Distributed methods for computing approximate equilibria. *Algorithmica*, 81(3):1205–1231, 2019.
- 15 Artur Czumaj, Michail Fasoulakis, and Marcin Jurdziński. Approximate well-supported Nash equilibria in symmetric bimatrix games. In *Proceedings of SAGT*, volume 8768, pages 244–254, 2014.
- 16 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. In *Proceedings of STOC*, pages 71–78, 2006.
- 17 Constantinos Daskalakis, Aranyak Mehta, and Christos Papadimitriou. Progress in approximate Nash equilibria. In *Proceedings of EC*, pages 355–358, 2007.
- 18 Constantinos Daskalakis, Aranyak Mehta, and Christos Papadimitriou. A note on approximate Nash equilibria. *Theoretical Computer Science*, 410(17):1581–1588, 2009.

- 19 Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. Approximating the existential theory of the reals. *J. Comput. Syst. Sci.*, 125:106–128, 2022. doi:10.1016/j.jcss.2021.11.002.
- 20 Argyrios Deligkas, John Fearnley, and Rahul Savani. Inapproximability results for constrained approximate Nash equilibria. *Inf. Comput.*, 262:40–56, 2018.
- 21 John Fearnley, Paul W. Goldberg, Rahul Savani, and Troels Bjerre Sørensen. Approximate well-supported Nash equilibria below two-thirds. *Algorithmica*, 76(2):297–319, 2016.
- 22 Ravi Kannan and Thorsten Theobald. Games of fixed rank: A hierarchy of bimatrix games. *Economic Theory*, 42(1):157–173, 2010.
- 23 Spyros C. Kontogiannis, Panagiota N. Panagopoulou, and Paul G. Spirakis. Polynomial algorithms for approximating Nash equilibria of bimatrix games. In *Proceedings of WINE*, pages 286–296, 2006.
- 24 Spyros C. Kontogiannis and Paul G. Spirakis. Well supported approximate equilibria in bimatrix games. *Algorithmica*, 57(4):653–667, 2010.
- 25 Spyros C. Kontogiannis and Paul G. Spirakis. Approximability of symmetric bimatrix games and related experiments. In *Proceedings of SEA*, volume 6630, pages 1–20, 2011.
- 26 Pravesh K. Kothari and Ruta Mehta. Sum-of-squares meets Nash: lower bounds for finding any equilibrium. In *Proceedings of STOC*, pages 1241–1248. ACM, 2018.
- 27 Richard Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *Proceedings of EC*, pages 36–41, 2003.
- 28 Zhengyang Liu and Ying Sheng. On the approximation of Nash equilibria in sparse win-lose games. In *Proceedings of AAAI*, volume 32(1), 2018.
- 29 Andrew McLennan and Rabee Tourky. Imitation games and computation. *Games and Economic Behavior*, 70(1):4–11, 2010.
- 30 Andrew McLennan and Rabee Tourky. Simple complexity from imitation games. *Games and Economic Behavior*, 68(2):683–688, 2010.
- 31 Ruta Mehta. Constant rank two-player games are PPAD-hard. *SIAM J. Comput.*, 47(5):1858–1887, 2018.
- 32 Aniket Murhekar and Ruta Mehta. Approximate Nash equilibria of imitation games: Algorithms and complexity. In *Proceedings of AAMAS*, pages 887–894, 2020.
- 33 John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.
- 34 Panagiota N. Panagopoulou and Paul G. Spirakis. Random bimatrix games are asymptotically easy to solve (a simple proof). *Theory of Computing Systems*, 54(3):479–490, 2014.
- 35 Aviad Rubinfeld. Settling the complexity of computing approximate two-player Nash equilibria. In *Proceedings of FOCS*, pages 258–265, 2016.
- 36 Haralambos Tsaknakis and Paul G. Spirakis. An optimization approach for approximate Nash equilibria. *Internet Mathematics*, 5(4):365–382, 2008.

Near Optimal Algorithm for Fault Tolerant Distance Oracle and Single Source Replacement Path Problem

Dipan Dey ✉

IIT Gandhinagar, India

Manoj Gupta ✉ 🏠

IIT Gandhinagar, India

Abstract

In a graph G with a source s , we design a distance oracle that can answer the following query: $\text{QUERY}(s, t, e)$ – find the length of shortest path from a fixed source s to any destination vertex t while avoiding any edge e . We design a deterministic algorithm that builds such an oracle in $\tilde{O}(m\sqrt{n})$ time¹. Our oracle uses $\tilde{O}(n\sqrt{n})$ space and can answer queries in $\tilde{O}(1)$ time. Our oracle is an improvement of the work of Bilò et al. (ESA 2021) in the preprocessing time, which constructs the first deterministic oracle for this problem in $\tilde{O}(m\sqrt{n} + n^2)$ time.

Using our distance oracle, we also solve the *single source replacement path problem* (SSRP problem). Chechik and Cohen (SODA 2019) designed a randomized combinatorial algorithm to solve the SSRP problem. The running time of their algorithm is $\tilde{O}(m\sqrt{n} + n^2)$. In this paper, we show that the SSRP problem can be solved in $\tilde{O}(m\sqrt{n} + |\mathcal{R}|)$ time, where \mathcal{R} is the output set of the SSRP problem in G . Our SSRP algorithm is optimal (upto polylogarithmic factor) as there is a conditional lower bound of $\Omega(m\sqrt{n})$ for any combinatorial algorithm that solves this problem.

2012 ACM Subject Classification Theory of computation; Theory of computation → Shortest paths; Theory of computation → Data structures design and analysis

Keywords and phrases distance sensitivity oracle, single-source replacement paths

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.42

Related Version *Full Version*: <https://arxiv.org/pdf/2206.15016.pdf>

1 Introduction

Real-life graph networks are prone to failures, e.g., nodes or links can fail. Thus, algorithms developed for these networks must be resilient to failures. For example, there may be some edges or links which are not working in the network and we want to avoid them. In this paper, we present an algorithm to create an oracle for the single source shortest path problem in a fault-prone graph. Such algorithms are also called fault-tolerant algorithms.

Consider an undirected and unweighted graph G with a source s . We want to build an oracle that can find the length of shortest path from s to any other vertex in the presence of faulty edges – such an oracle is also called a *fault-tolerant distance oracle*. Formally,

► **Definition 1.** *A fault-tolerant distance oracle answers the following query in a graph G :*

$\text{QUERY}(s, t, F)$: *Find the length of shortest path from s to t avoiding the set F of edges.*

¹ $\tilde{O}()$ hides polylog n factor



The time it takes to answer a query is called the *query time*. If the query is always from a fixed source s and $|F| \leq f$, then the distance oracle is called a f -edge fault tolerant single source distance oracle, or $\text{SDO}(f)$ in short. If all vertices can be sources, the oracle is called f -edge fault tolerant distance oracle, or $\text{DO}(f)$. We list some results related to distance oracles:

Demetrescu et al. [12] designed a $\text{DO}(1)$ with $\tilde{O}(n^2)$ space and $O(1)$ query time. Bernstein and Karger[3] showed that this oracle can be built in $\tilde{O}(mn)$ time. Pettie and Duan [15] extended the result of Demetrescu et al. to two faults. They designed a $\text{DO}(2)$ with $\tilde{O}(n^2)$ space and $\tilde{O}(1)$ query time. Gupta and Singh [20] designed a $\text{SDO}(1)$ with $\tilde{O}(n\sqrt{n})$ space and $\tilde{O}(1)$ query time. Recently Bilò et al. [5] built the $\text{SDO}(1)$ (described in [20]) in $\tilde{O}(m\sqrt{n} + n^2)$ time. They also showed that, the lower bound of size for such an oracle is $O(n\sqrt{n})$. Many different aspects of distance oracles have been studied in literature [4, 3, 6, 9, 10, 14, 29]. In the line of k simple shortest paths the some important works are [27, 28, 2]

In this paper, we will focus our attention on building $\text{SDO}(1)$. Due to Bilò et al. [5], the time to build $\text{SDO}(1)$ is $\tilde{O}(m\sqrt{n} + n^2)$. Chechik and Cohen [8] showed that, the first term in this running time is a conditional lower bound for SSRP problem. But it is not clear if the second term is necessary. In this paper, we build a $\text{SDO}(1)$ in $\tilde{O}(m\sqrt{n})$ time – this preprocessing algorithm has a better runtime than [5] for sparse graphs, which is state of the art for this problem till now. Using our $\text{SDO}(1)$ data structure, we are able to reduce the runtime of the algorithm solving SSRP problem too. Our distance oracle is quite different from the distance oracle of Gupta and Singh [20] – though we use the main technical idea of [20] crucially in our paper too. The construction of this new oracle is the main technical result of this paper.

► **Theorem 2.** *For undirected, unweighted graphs there is a deterministic algorithm that can build a $\text{SDO}(1)$ of size $\tilde{O}(n\sqrt{n})$ and query time $\tilde{O}(1)$ in $\tilde{O}(m\sqrt{n})$ time.*

1.1 Application: Single Source Replacement Path Problem

Let us first look at the *replacement path problem*. In this problem, we are given a source s and a destination t . We assume that there is a unique shortest path from s to t , denoted by st .

► **Definition 3** (Replacement Path Problem). *Let s be a source and vertex t be the destination in G . For each $e \in st$ path, output the length of the shortest path from s to t avoiding e .*

The replacement path problem was first investigated due to its relation with auction theory [21, 24] and has been studied extensively. For an undirected graph with non-negative edge weights, the replacement path problem can be solved in $\tilde{O}(m + n)$ time[22, 21, 23]. We look at the generalization of the replacement path problem – the single source replacement path problem.

► **Definition 4** (SSRP problem). *Let s be a source in a graph G which is undirected and unweighted. For each vertex $t \in G$ and each $e \in st$ path, output the length of the shortest path from s to t avoiding e .*

Chechik and Cohen [8] designed a randomized combinatorial algorithm that solves the SSRP problem in $\tilde{O}(m\sqrt{n} + n^2)$ time. They also showed a matching conditional lower bound via Boolean Matrix Multiplication.

► **Lemma 5** ([8]). *Let $BMM(n, m)$ be the time taken to multiply two $n \times n$ boolean matrices with a total of m ones. Under the assumption that any combinatorial algorithm for $BMM(n, m)$ requires $mn^{1-o(1)}$ time², any combinatorial algorithm for SSRP problem requires $\Omega(m\sqrt{n})$ time.*

It may seem that the algorithm of Chechik and Cohen [8] is nearly optimal. It is indeed the case if the output size is $O(n^2)$. However, for a low-diameter graph, this extra additive factor seems unnecessary. If the graph is dense ($m \geq n^{3/2}$), then the n^2 factor is subsumed by the first term $m\sqrt{n}$. Thus, when $m < n^{3/2}$ and the graph has a low diameter, can we improve the running time of the SSRP problem? For such a graph, the algorithm of Chechik and Cohen [8] is not optimal. Similar to [8], Gupta et al. [18] also designed an algorithm for the SSRP problem. Even this algorithm has the running time $\tilde{O}(m\sqrt{n} + n^2)$ – though it uses an entirely different approach compared to [8]. Thus, the main question is: *can we remove this extra additive factor of n^2 from the running time of the SSRP problem?* In this paper, we design such an algorithm:

► **Theorem 6.** *There is a deterministic algorithm for SSRP problem with a running time of $\tilde{O}(m\sqrt{n} + |\mathcal{R}|)$ where $|\mathcal{R}|$ is the output size of SSRP problem in G .*

In the above theorem, $|\mathcal{R}|$ is the output size, thus an implicit lower bound on the SSRP problem. Using Lemma 5, we conclude that our algorithm is nearly optimal up to a polylogarithmic factor.

To build an algorithm for the SSRP problem, we first build a SDO(1). Then, for each $t \in G$ and each $e \in st$ path, we call $\text{QUERY}(s, t, e)$ and output the answer. Thus, we claim the following lemma:

► **Lemma 7.** *If we can build a SDO(1) with query time q in time T , then there is an algorithm for SSRP problem with a running time $O(T + q|\mathcal{R}|)$, where $|\mathcal{R}|$ is the output size in G .*

The above lemma, along with Theorem 2 implies Theorem 6.

1.2 Related Work

Other related problems include the fault-tolerant subgraph problem. In this problem, we want to find a subgraph of G such that the shortest path from s is preserved in the subgraph after any edge deletion. Parter and Peleg [26] designed an algorithm to compute a single fault-tolerant subgraph with $O(n^{3/2})$ edges. They also showed that their result could be easily extended to multiple sources. This result was later extended to dual fault by Parter [16] with $O(n^{5/3})$ edges. Gupta and Khan [19] extended the above result to multiple sources. All the above results are optimal due to a result by Parter [25] which states that a multiple source f -fault tolerant subgraph requires $\Omega\left(n^{2-\frac{1}{f+1}}\right)$ edges. Bodwin et al. [7] showed the existence of a f -fault tolerant subgraph of size $O\left(fn^{2-\frac{1}{2f}}\right)$.

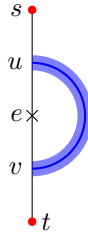
2 Preliminaries

Let $G(V, E)$ be an undirected unweighted graph with a source s . Given two vertices u and v in a graph H , unless otherwise stated, $(uv)_H$ denotes the shortest path from u to v in H . If $H = G$, we will remove the subscript and the brackets – we will apply this policy for all the

² In a RAM model with words of $O(\log n)$ bits.

notations below. $|uv|_H$ denotes the length of the shortest path in H . Some of our graphs will be weighted, even though G is unweighted. If H is weighted, then we will abuse notation and use $|uv|_H$ to denote the weight of the shortest path from u to v in H . The edges and vertices of H will be denoted by E_H and V_H , respectively. Additionally, m_H and n_H will denote the number of edges and vertices in H , respectively. $\text{SPT}_H(s)$ denotes the shortest path tree from s in H . We can view the $\text{SPT}_H(s)$ to be drawn from top to bottom with the top vertex being s . For any two vertices u, v on the st path in $\text{SPT}_H(s)$, we say that u is before / above v if $|su|_H < |sv|_H$. Similarly, we say that u is after/below v if $|su|_H > |sv|_H$. For an edge e in a weighted graph H , $wt_H(e)$ will denote the weight of e . Given two paths $(uw)_H$ and $(vw)_H$, the path $(uw)_H + (vw)_H$ denotes their concatenation. $P[u, v]$ denotes a contiguous subpath of P starting at u and ending at v . Sometimes, we may also write *the interval* $[u, v]$ of P to denote $P[u, v]$. We say u comes before v on a path R starting from s , if $|R[s, u]| < |R[s, v]|$. Similarly, we can define the term u comes after v on path R .

A replacement path R is the shortest path from s to t avoiding an edge e on st path. There can be many replacement paths of the same length avoiding e . To ensure uniqueness, we will use the following definition of replacement path³.



■ **Figure 1** Replacement path with detour uv and detour point u .

► **Definition 8 (Replacement Path).** A path R from s to t avoiding e is called a replacement path if (1) it diverges from and merges to the st path just once (2) its divergence point from the st path is as close to s as possible. (3) it is the **lexicographically smallest**⁴ shortest path in G satisfying (1) and (2).

We now define some terms related to replacement paths. $(st \diamond e)_H$ denotes the replacement path from s to t avoiding the edge e in H . We can generalize this notation to a replacement path that avoids a set of edges. Thus, $(st \diamond F)_H$ denotes the replacement path from s to t in H avoiding a set F of edges. In our algorithm, after we find the replacement path $(st \diamond e)_H$, we will store its length in $d_H(s, t, e)$. Sometimes, we also want to store the length of $(st \diamond F)_H$. In that case, we will store it in $d_H(s, t, F)$.

► **Definition 9 (Detour and Detour point of a replacement path).** Let $R = st \diamond e$. Then, the detour of R is $R \setminus st$. That is, let us assume that R leaves st above e at a vertex u , and merges back on st at vertex v after e , then detour of R is $R[u, v]$. Also, the vertex at which the detour starts is called the detour point of R . So, u is the detour point of R or in short $DP(R) = u$.

³ This was referred to as *preferred replacement path* in [19].

⁴ Let P and P' first diverge from each other to $x \in P$ and $x' \in P'$ respectively. If the index of x is lower than x' , then P is said to be lexicographically smaller than P' .

Lastly, in our algorithm, we will need to find least common ancestor of any two vertices u and v in $\text{SPT}_H(s)$. Let $\text{LCA}_H(u, v)$ denotes the least common ancestor of u and v in $\text{SPT}_H(s)$. To find the LCA, we will use the following result:

► **Lemma 10** (See [1] and its references). *Given a tree T on n vertices, we can build a data structure of size $O(n)$ in $O(n)$ time such that the least common ancestor query can be answered in $O(1)$ time.*

3 Overview of our algorithm to build SDO(1)

We will use divide and conquer approach to build SDO(1). This strategy has been previously used for directed graphs in [16, 11]. However, simply using this strategy will not get us close to our desired bound of $\tilde{O}(m\sqrt{n})$. For that, we need to combine this divide and conquer strategy with an idea of Gupta and Singh [20]. This combination is one of the technical contributions of the paper.

Like [16, 11], we use the following separator lemma to divide the graph G .

► **Theorem 11** (Separator Lemma [16, 11]). *Given a tree T with n nodes rooted at a source s , one can find in $O(n)$ time a vertex r that separates the tree T into edge disjoint sub-trees M, N such that $E_M \cup E_N = E_T, V_M \cap V_N = \{r\}$ and $\frac{n}{3} \leq |V_M|, |V_N| \leq \frac{2n}{3}$.*

Without loss of generality, we will assume that $s \in V_M$. Thus, r is the root of N . Also, note that s and r may or may not be the same. Let G_M and G_N be the graph induced by the vertices of M and N , respectively. There is one more important term that we will use in our paper:

► **Definition 12** (Primary Path \mathcal{P}). *Using the separator lemma, $\text{SPT}(s)$ can be divided into two sub-trees M and N with roots s and r . The path from s to r is called the primary path and is denoted by \mathcal{P} .*

We now describe our data structure that we will build recursively. We can view the data structure as a binary tree \mathcal{T} . The root contains data structure for the entire graph G . We will abuse notation and say the root is the graph G .

The left child of the root will contain the graph G_M and some weighted edges – we will describe the utility of these weighted edges in the next section. These weighted edges are not in G but are added by our algorithm. We will then build a data structure for G_M recursively. The right child of the root will contain the graph G_N , again with some weighted edges.

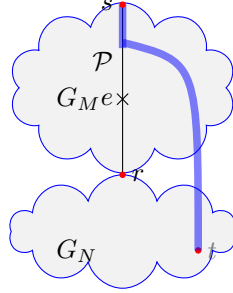
At the root of \mathcal{T} , we store the following data structures. For each $v \in G$, set $d(s, v) = |sv|$. Similarly, set $d(r, v) = |rv|$. For each $v \in G_N$, set $d(s, v, G_N) = |sv \diamond G_N|$. Similarly, for each $v \in G_M$, set $d(s, v, G_M) = |sv \diamond G_M|$. All these quantities can be computed using a single source shortest path algorithm in $\tilde{O}(m+n)$ time. Additionally, we will find the length of all replacement paths from s to r avoiding edges on the primary path \mathcal{P} . This can be done in $\tilde{O}(m+n)$ time using⁵ [21]. We will set $d(s, r, e) = |sr \diamond e|$ for each edge $e \in \mathcal{P}$.

We store the above data-structure in each node of \mathcal{T} . If a node of \mathcal{T} contains graph H , then we can construct the above data-structures in $\tilde{O}(m_H + n_H)$ time. We now describe our algorithm that finds replacement paths using \mathcal{T} .

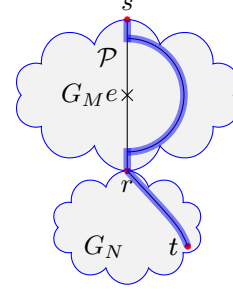
Let us see how we find and store lengths of the replacement paths at the root of \mathcal{T} , that contains graph G . First, we find the replacement paths for edges on the primary path. Let $R = st \diamond e$ where $e \in \mathcal{P}$. We define R to be either *jumping* or *departing* depending on whether it merges back to the primary path or not.

⁵ This algorithm work for graphs with non-negative edge weights. And our graph may have weighted edges.

► **Definition 13** (Jumping and Departing paths [11, 17]). Let $R = st \diamond e$ where $e \in \mathcal{P}$. R is called a **jumping path** if it uses some vertex $u \in \mathcal{P}$ after e . If the path is not jumping then it is a **departing path**. If a replacement path is jumping, then it is called jumping replacement path. Similarly, we define departing replacement path. See Figure 2 for a visualization of these two kinds of paths.



(a) Departing Replacement Path.



(b) Jumping Replacement Path.

■ **Figure 2** Departing and Jumping Replacement Path.

Note that, jumping or departing path is defined only when the edge fault is on the primary path. Also, if a replacement path is departing, then the destination t cannot lie on \mathcal{P} . In Section 4, we will find all jumping replacement paths.

In Section 5, we design a new algorithm for finding and storing all departing replacement paths. To this end, we will use the main idea in the paper of Gupta and Singh [20]. In [20], the authors sampled a set of vertices with probability of $O(\frac{1}{\sqrt{n}})$. Then, for a vertex $t \in G$, they find a sampled vertex near t on the st path. They call this vertex t_s . Then, they show the following important lemma, which is the main idea of their paper:

► **Lemma 14** (Lemma 11 in [20]). *The number of replacement paths from s to t that avoid edges in st_s path and also avoid t_s is $O(\sqrt{n})$.*

An astute reader can see that the definition of replacement paths in the above definition looks very similar to departing replacement paths. We prove that this is indeed the case. Thus, we can transfer the result in Lemma 14 to departing replacement paths. This is the main novelty of the paper. The main technical result of Section 5 is as follows:

► **Lemma 15.** *For each $t \in G$, all departing replacement paths to t can be found in deterministic $\tilde{O}(m\sqrt{n})$ time. Moreover, the length of all such departing paths can be stored in a data structure of size $\tilde{O}(n\sqrt{n})$ and can be queried in $\tilde{O}(1)$ time.*

4 Algorithm to build SDO(1): Replacement paths that are not departing

In Section 5, we will find all and store all departing replacement paths. Thus, we just need to concentrate on the replacement paths that are either jumping or the faulty edge $e \notin \mathcal{P}$. We now divide remaining replacement paths depending on where the destination t and faulty edge e lies. There are following cases:



■ **Figure 3** $e \in G_M$ and $t \in G_N$.

4.1 $e \in G_M$ and $t \in G_N$

This case itself can be divided into two cases depending on whether e lies on the primary path or not.

1. $e \in \mathcal{P}$ (See Figure 3(a))

Let $R = st \diamond e$. If R is departing then we will see how to find it in Section 5. So, assume that R is jumping. This implies that R merges back to \mathcal{P} at a vertex, say w , after the edge e . Since $t \in G_N$, $st = sw + wr + rt$. Thus, after merging with \mathcal{P} at w , the replacement path passes through r . In that case, $|st \diamond e| = |sr \diamond e| + |rt|$. We can easily find the right hand side of the above equality as we have stored $d(s, r, e) = |sr \diamond e|$ and $d(r, t) = |rt|$.

2. $e \notin \mathcal{P}$ (See Figure 3(b))

In this case, we claim that $st \diamond e = st$. The st path has \mathcal{P} as its prefix. Since \mathcal{P} lies in G_M and survives after the deletion of e , st path remains intact.

4.2 $e \in G_M$ and $t \in G_M$

Since both e and t lie in G_M , one may think that we can recurse our algorithm in G_M to find $st \diamond e$. If $st \diamond e$ completely lies inside G_M , this is indeed the case. However, $st \diamond e$ may also use edges of G_N . To handle such cases, before recursing in G_M , we will add weighted edges to it. For each $v \in G_M$, we will add an edge from r to v with a weight $|rv \diamond G_M|$. We have already calculated this weight, it is stored in $d(r, v, G_M)$. Let the set of weighted edges added to G_M be called X . We now look at two cases, (1) $e \in \mathcal{P}$ and (2) $e \notin \mathcal{P}$.

4.2.1 $e \in \mathcal{P}$

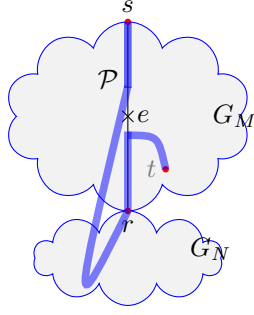
Let $R = st \diamond e$ be a jumping replacement path. We will show that $st \diamond e = sr \diamond e + rt$. As we have calculated the length of both the paths in the right-hand side of the above equality, there is no need to even recurse in this case. To prove the above equality, we first prove the following simple lemma:

► **Lemma 16.** *Let $e \in \mathcal{P}$, $t \in G_M$. Assume that the jumping replacement path $R = st \diamond e$ uses some edges of G_N . Then $st \diamond e$ passes through r .*

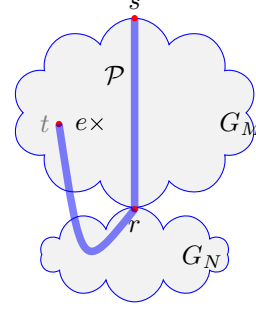
Proof. Since R is jumping, it merges with \mathcal{P} . There are two ways in which R can merge with \mathcal{P} .

1. R merges with \mathcal{P} and then visits the edges of G_N .

Let us assume that u is the last vertex of G_N in the path R and R merges with \mathcal{P} at w . Since R first merges with \mathcal{P} and then visits the edges of G_N , u comes after w on R .



(a) $e \in \mathcal{P}, t \in G_M$ and the jumping replacement path uses G_N .



(b) $e \in G_M \setminus \mathcal{P}, t \in G_M$ and the replacement path uses G_N .

■ **Figure 4** $e \in G_M$ and $t \in G_M$.

Since w is below e on \mathcal{P} , we claim that the sub-path wu of su survives in $G \setminus e$ and is also the shortest path from w to u . But wu path passes through r . Thus, $st \diamond e$ passes through r by construction in Theorem 11.

2. R visits an edge of G_N and then merges with \mathcal{P} (See Figure 4(a))

In this case, we will show that R merges with \mathcal{P} at r . For contradiction, let w be the vertex at which R merges with \mathcal{P} such that $w \neq r$. Let u be the first vertex of G_N visited by R . Also, w lies after u on path R . Thus, the replacement path $R = R[s, u] + R[u, w] + R[w, t]$. Since w lies below e on \mathcal{P} , wu sub-path of su does not contain e and is also the shortest path from w to u . Thus, $R[u, w] = uw$. But uw path passes through r . This implies that R merges with \mathcal{P} at r contradicting our assumption that $w \neq r$. ◀

We are now ready to prove the main lemma in this subsection.

► **Lemma 17.** *Let $e \in \mathcal{P}$ and $t \in G_M$. Assume that the jumping replacement path $R = st \diamond e$ uses some edges of G_N . Then $|st \diamond e| = |sr \diamond e| + |rt|$.*

Proof. Using Lemma 16, R passes through r . So, we have $R = R[s, r] + R[r, t]$. The first summand on the right hand side of the above equality represent a path from s to r avoiding e . Thus, $|R[s, r]| = |sr \diamond e|$.

We will now show that $|R[r, t]| = |rt|$. Clearly $|R[r, t]| \geq |rt|$ as the first path avoid e and the second path may or may not. We will now show that the second path also avoids e which will imply that both paths are of same length. For contradiction, assume that rt passes through e . Let us assume that there is a vertex w before the edge e on path \mathcal{P} such that $rt = rw + wt$. Thus, rw passes through e but wt avoids e . But then, there is a path R' such that $R' = sw + wt$ which avoids e . We claim that $|R'| < |R|$ contradicting the fact that R is the replacement path from s to t avoiding e .

To this end,

$$\begin{aligned} |R| &= |sr \diamond e| + |R[r, t]| \\ &\geq |sr \diamond e| + |rt| \\ &= |sr \diamond e| + |rw| + |wt| \\ &\geq |sr \diamond e| + |wt| \end{aligned}$$

$$\begin{aligned} \text{Since } |sw| < |sr|, |sw| &< |sr \diamond e| \\ &> |sw| + |wt| \\ &= |R'|. \end{aligned}$$

This completes the proof of the lemma. ◀

4.2.2 $e \notin \mathcal{P}$

In this case, we will show a path in $G_M \cup X$ such that it avoids e and has the same length as $st \diamond e$. Please see Figure 4(b) for a visualization of this case.

► **Lemma 18.** *Let $e \in G_M \setminus \mathcal{P}$ and $t \in G_M$. Assume that the replacement path $R = st \diamond e$ uses some edges of G_N . Then there is a path in $G_M \cup X$ that avoids e and has length $|st \diamond e|$.*

Proof. Let us first prove that R can alternate between edges of G_N and G_M just once. To this end, let u be the last vertex of G_N visited by R . Thus, $R = R[s, u] + R[u, t]$. But the shortest path su remains intact in $G \setminus e$ as $e \notin \mathcal{P}$ and the shortest su path passes through r . This implies that $R = R[s, u] + R[u, t] = R[s, r] + R[r, u] + R[u, t]$. By construction, the first path $R[s, r] = sr$ and it completely lies in G_M . The second path $R[r, u] = ru$ and it completely lies in G_N . Let v be the vertex just after u in R . So, $v \in G_M$. So we have $R = sr + ru + (u, v) + R[v, t]$. By construction, $R[v, t]$ completely lies in G_M . Thus, R alternates from edges of G_N to G_M just once.

From the above discussion $R = R[s, r] + R[r, u] + (u, v) + R[v, t] = R[s, r] + R[r, v] + R[v, t]$. The first and the last paths of the above equality completely lies in G_M . By construction, $R[r, v]$ does not contain any edge of G_M . Thus, $R[r, v]$ is the shortest path from r to v avoiding edges of G_M , that is $|R[r, v]| = |rv \diamond G_M| = d(r, v, G_M)$.

Since we have added an edge from r to v with weight $d(r, v, G_M)$, we will now show that there is a path in $G_M \cup X$ that avoids e and has same weight as R . Consider the path $R' = R[s, r] + (r, v) + R[v, t]$. The reader can check all the three subpaths lie completely in $G_M \cup X$. Moreover, $(r, v) \in X$ is a weighted edge. Thus the weight of $|R'| = |R[s, r]| + wt_{(G_M \cup X)}(r, v) + |R[v, t]| = |R[s, r]| + d(r, v, G_M) + |R[v, t]| = |R[s, r]| + |R[r, v]| + |R[v, t]| = |R|$. This completes the proof. ◀

4.3 $e \in G_N$ and $t \in G_M$

In this case, st path completely lies in G_M and thus survives. Thus, $|st \diamond e| = |st| = d(s, t)$.

4.4 $e \in G_N$ and $t \in G_N$

In this case, we will recurse in G_N . However, G_N may not contain the source s if $s \neq r$. In that case, before recursing, we add a new source s_N in G_N . We also add some *weighted* edges to G_N . For each $v \in G_N$, we add an edge from s_N to v with a weight $d(s, v, G_N) = |sv \diamond G_N|$. Let this set of edges be called Y . Let us now show that we will find all the replacement paths if we recurse in $G_N \cup Y$.

► **Lemma 19.** *Let $e \in G_N$ and $t \in G_N$. There is a path from s_N to t in $G_N \cup Y$ that avoids e and has weight $|st \diamond e|$.*

Proof. Let $R = st \diamond e$. Let us first prove that once R visits an edge of G_N , it cannot visit an edge of G_M anymore. Let u be the last vertex of G_M visited by R and v be the vertex after u in R . Thus, $v \in G_N$. Thus, $R = R[s, u] + (u, v) + R[v, t]$. By construction, $R[v, t]$ completely lies in G_N . The shortest path su remains intact in $G \setminus e$ as $e \in G_N$ and the path su completely lies in G_M . Thus, $R[s, u] = su$. So, the path R first visits only edges of G_M (in $R[s, u]$), then goes to G_N (by taking the edge (u, v)) and then remains in G_N (in $R[v, t]$). Thus, R does not visit any edge of G_M once it visits an edge of G_N .

By the above discussion, $R = R[s, u] + (u, v) + R[v, t] = R[s, v] + R[v, t]$ such that $R[v, t]$ completely lies in G_N . Also, $R[s, v]$ completely lies in G_M except the last edge which has one endpoint in G_M and other endpoint $v \in G_N$. Thus, $R[s, v] = |sv \diamond G_N| = d(s, v, G_N)$ and we have $R = sv \diamond G_N + R[v, t]$.

Now consider a path R' that avoids e from s_N to t in $G_N \cup Y$. We construct this path as follows: we will first take the weighted edge $s_N \rightarrow v$ and then the path $R[v, t]$. The reader can check that these two subpaths completely lie in $G_N \cup Y$. The weight of this path is $|R'| = |s_N \diamond G_N| + |R[v, t]| = d(s, v, G_N) + |R[v, t]| = |R[s, v]| + |R[v, t]| = |R|$. ◀

4.5 One endpoint of e is in G_M and the other is in G_N and $t \in G$

This is an easy case as the st path survives in $G \setminus e$ as st does not contain e . Thus, $st \diamond e = st$

5 Departing Replacement paths

In the previous section, we have already found out a replacement path if it is jumping or if the edge failure is not on the primary path. In this section, we will try to find the best departing path after an edge failure. To this end, we define the following:

► **Definition 20** (Candidate departing paths). *Let $e \in \mathcal{P}$ and let D be the set of all paths avoiding e that do not use any vertex of \mathcal{P} after e . $P \in D$ is called the candidate departing path for e , if among paths in D , P has the minimum length. In case there are many departing paths avoiding same edge with same length, we will break ties using Definition 8.*

Note that, a candidate departing path may or may not be a replacement path. In case it is, we call it a departing replacement path. Also, P may be a candidate departing path for all edges in an interval, say $yz \in \mathcal{P}$, but may be a replacement path for a sub-interval of yz . With this definition in hand, we will now find all candidate departing paths.

5.1 Finding all candidate departing paths

In the previous section, we added some weighted edges in the graph when we recurse. Thus, there might be two types of edges in the graph – *weighted* and *unweighted*. The weighted edges represent paths in the graph G , and the unweighted edges are present in G . G contains only unweighted edges. However, the graph at an internal node of \mathcal{T} , say graph \widehat{G} , may have weighted as well as unweighted edges.

In the ensuing discussion, let \widehat{G} be the graph processed by our algorithm at some internal node of \mathcal{T} . In the graph \widehat{G} , we will assume that there is a source s . Let \overline{G} be the parent of \widehat{G} . In our algorithm, we partition \overline{G} into two disjoint graphs and then recurse on it. If \widehat{G} is the left child of \overline{G} , then it includes a set X of weighted edges added by us in Section 4.2. Similarly, if \widehat{G} is the right child of \overline{G} then it includes a set Y of weighted edges added by us in Section 4.4. Using the separator lemma, we will find the vertex r that partitions $\text{SPT}_{\widehat{G}}(s)$. Also, the primary path $\mathcal{P} = sr$.

We now give an overview of our method to find candidate departing paths in \widehat{G} . To this end, we will use the main idea in the paper of Gupta and Singh [20]. In [20], the authors proved Lemma 14. Though they did not mention it, the paths in the Lemma 14 look very much like the departing paths. Indeed, that lemma is more general than what the authors originally intended it to be. The authors show the above lemma for a specific vertex t_s , but a careful reading of the paper suggests that the above lemma is true for any vertex on the st path. We now generalise this lemma. However, there is one problem. The above result holds only for an unweighted graph, whereas \widehat{G} is weighted. Thus, we cannot prove the above lemma as it is. However, we will prove the following weaker lemma:

► **Lemma 21.** *Let \widehat{G} be the graph at an internal node in the binary tree \mathcal{T} . Let s be the source of \widehat{G} . For a destination $t \in \widehat{G}$, let p be any vertex on st path. The number of replacement paths from s to t that avoid edges on sp path and also avoid p is $O(\sqrt{n})$.*

Some discussion is in order. In an ideal case, the number of replacement paths that avoid edges on sp and also avoid p should have been $O(\sqrt{n_{\widehat{G}}})$. This result would have been similar to Lemma 14. Unfortunately, we cannot prove this result as \widehat{G} is weighted. However, if we just expand the weighted edge in the graph \widehat{G} , then we will get an unweighted graph. By expanding, we mean that for each weighted edge, add the path that represents that weighted edge. However, this process may increase the number of vertices in the graph to n . Now, we can adapt Lemma 14. For an unweighted graph with n vertices, this lemma guarantees that the number of replacement paths avoiding p will be at most \sqrt{n} . Indeed, this is our result in Lemma 21. Interested reader can find the proof of this in the full version of the paper [13] as it is an extension of the proof in [20].

Lemma 21 implies that there are only $O(\sqrt{n})$ replacement paths that have some special properties which are similar to the properties of departing paths. So, our plan of action is as follows:

1. Show that there are only $O(\sqrt{n})$ candidate departing paths to t in \widehat{G} . This will be done by showing the similarity between the replacement paths in Lemma 21 and candidate departing paths.
2. Show that we can find the lengths of all the candidate departing paths in \widehat{G} in $O(m_{\widehat{G}}\sqrt{n})$ time. Additionally, we show that we can store the lengths in a compact data structure of size $O(n_{\widehat{G}}\sqrt{n})$. Given any query $\text{QUERY}(s, t, e)$ to this data-structure such that $s, t \in \widehat{G}$ and e is on the primary path of \widehat{G} , we can find the length of corresponding candidate departing path in $\widetilde{O}(1)$ time.

This completes the overview of our algorithm for finding candidate departing paths.

5.2 Similarity between candidate departing paths and replacement paths in Lemma 21

Let us first prove some simple results that will help us prove this section's main idea.

► **Lemma 22.** *Let $t \in \widehat{G}$ and $p = \text{LCA}(t, r)$. All the candidate departing paths from s to t avoiding edges on sp path also avoid p . For any edge e on pr path, $st \diamond e = st$.*

Proof. Let $R = st \diamond e$ where $e \in sp$ and R is departing. The detour of R must start above e on \mathcal{P} . Since R is departing, it can not merge with the path \mathcal{P} again. So, it avoids p also.

The st path passes through p and does not use any vertex of \mathcal{P} below p . So, removing any edge on pr path does not disturb st path. So, for any edge $e \in pr$, $st \diamond e = st$. ◀

We now show that the candidate departing paths and the replacement paths in Lemma 21 have the same property.

► **Lemma 23.** *For any vertex $t \in \widehat{G}$, there are $O(\sqrt{n})$ candidate departing paths to t .*

Proof. The proof of the lemma can be derived using Lemma 22 and Lemma 21 together. For complete proof please see the full version [13]. ◀

Given the above lemma, we need to store $O(\sqrt{n})$ candidate departing paths to t in \widehat{G} . Before designing an algorithm to find candidate departing paths, we first see how we plan to store these paths in a compact data structure.

5.3 The data-structure at each node of \mathcal{T}

Let us first discuss a small technical detail that may be perceived as a problem but is not. Our graph \widehat{G} is weighted. It stands to reason that even the primary path \mathcal{P} in \widehat{G} maybe weighted. Since candidate departing paths are only for the faults on the primary path, it may not be clear what happens if the edge on the primary path is weighted. To this end, we show that on any st path in \widehat{G} , all edges except may be the first one, are unweighted.

► **Lemma 24.** *For $t \in \widehat{G}$, except may be the first edge of $(st)_{\widehat{G}}$, all other edges of $(st)_{\widehat{G}}$ are unweighted.*

Proof. We will prove using induction on the nodes of \mathcal{T} . In the root of \mathcal{T} , we have the graph G . Clearly, the graph G satisfies the property of the lemma. Let us assume using induction that the parent of the graph \widehat{G} satisfies the property of the lemma. Let \overline{G} be the parent of \widehat{G} . Let \bar{s} be the source in \overline{G} . Thus, in the graph \overline{G} , using the separator lemma, we find a \bar{r} that divides \overline{G} into two parts. The path from \bar{s} to \bar{r} , say $\overline{\mathcal{P}}$ is the primary path. By induction, only the first edge of the primary path may be weighted. There are two cases:

1. \widehat{G} is the left child of \overline{G} .

In this case, the source of \widehat{G} is also \bar{s} . For each $t \in \widehat{G}$, it can be observed that the path $(\bar{s}t)_{\overline{G}} = (\bar{s}t)_{\widehat{G}}$. Using induction hypothesis, since $(\bar{s}t)_{\overline{G}}$ satisfies the statement of lemma, so does $(\bar{s}t)_{\widehat{G}}$.

2. \widehat{G} is the right child of \overline{G} .

There are two cases. When $\bar{s} = \bar{r}$, then we fall back in point(1). So, let us look at the case when $\bar{s} \neq \bar{r}$.

For a $t \in \widehat{G}$, by construction, $(\bar{s}t)_{\overline{G}}$ passes through \bar{r} . Thus, $(\bar{s}t)_{\overline{G}} = (\bar{s}\bar{r})_{\overline{G}} + (\bar{r}t)_{\overline{G}}$. In \widehat{G} , we add a new source s . Also, we add a weighted edge from s to \bar{r} in \widehat{G} . The weight of this edge is $|\bar{s}\bar{r} \diamond \widehat{G}|_{\overline{G}}$. Also, $(\bar{r}t)_{\overline{G}} = (\bar{r}t)_{\widehat{G}}$. This implies that there is path in \widehat{G} from s to t , $(s, \bar{r}) + (\bar{r}t)_{\widehat{G}}$. By induction, we claim that on this path except (s, \bar{r}) , all the edges are unweighted. ◀

Using the above lemma, all except the first edge of the primary path are unweighted. The weighted edges represent edges for which we have already found candidate departing paths at the parent or an ancestor of \widehat{G} . Thus, we will only find candidate departing paths for unweighted edges in \mathcal{P} . In the ensuing discussion, whenever we mention a path avoiding an edge on the primary path, it will always refer to an unweighted edge.

We now prove some simple lemmas that will help us build data structures for candidate departing paths.

► **Lemma 25.** *Let R and R' be two different candidate departing paths from s to t avoiding edges e and e' respectively on the path \mathcal{P} . If e lies above e' on \mathcal{P} , then $|R| > |R'|$.*

Proof. The detour of the candidate departing R starts before e , and once it departs, it does not merge with \mathcal{P} again. As e lies above e' on \mathcal{P} , R also avoids e' . If $|R| \leq |R'|$, then by Definition 8, R must be the candidate departing path avoiding e' , leading to a contradiction. So, it must be the case that $|R| > |R'|$. ◀

► **Lemma 26.** *Let R be a candidate departing path. Let yz be the maximal subpath of \mathcal{P} such that R is the candidate departing path for edges in yz . Then $DP(R) = y$.*

Proof. For complete proof of the lemma please see the full version [13]. ◀

The above lemma states that if R avoids edges in yz , then the detour of R necessarily starts from y . We will now prove the contrapositive of the Lemma 25.

► **Lemma 27.** *Let R and R' be two different candidate departing paths from s to t . If $|R| > |R'|$, then $DP(R)$ lies above $DP(R')$ on \mathcal{P} .*

Proof. For complete proof of the lemma please see the full version [13]. ◀

We will now use the above lemmas and our deduction to build a compact data-structure for all candidate departing paths. To this end, we will store an array $DEP(t)$ for each $t \in \widehat{G}$. $DEP(t)$ will store candidate departing paths from s to t in increasing order of their lengths. By Lemma 23, there are $O(\sqrt{n})$ such paths. let us denote them by R_1, R_2, \dots, R_k where $k = O(\sqrt{n})$. For any two consecutive candidate departing paths R_i and R_{i+1} , using Lemma 27 and Lemma 26, we claim that R_{i+1} is the candidate departing path avoiding edges in $[DP(R_{i+1}), DP(R_i)]$ on the primary path \mathcal{P} . Since the size of $DEP(t) = O(\sqrt{n})$, the total size of DEP data-structure is bounded by $O(n_{\widehat{G}}\sqrt{n})$.

► **Lemma 28.** $\sum_{t \in \widehat{G}} \text{size of } DEP(t) = O(n_{\widehat{G}}\sqrt{n})$

5.4 Finding and storing all candidate departing paths efficiently

Let us first describe the setting that will be used throughout this section. At an internal node \widehat{G} of \mathcal{T} , we are planning to find all candidate departing paths from the source s . To this end, we will find a vertex r that will divide $SPT_{\widehat{G}}(s)$ roughly equally. Also, $\mathcal{P} = sr$.

To find all candidate departing path, we will build an auxiliary graph G which we will build incrementally. The source in this graph is (s) . All other vertices in G are tuples of the form $(v, |R|)$, where $v \in \widehat{G} \setminus \mathcal{P}$ and R is a candidate departing path to v . During initialization, we will add $(v, |sv|_{\widehat{G}})$ in G for each $v \in \widehat{G} \setminus \mathcal{P}$. Also, there will be an edge from (s) to $(v, |sv|_{\widehat{G}})$ with weight $|sv|_{\widehat{G}}$. We will show the following property at the end of our analysis.

► **Property 29.** *Let R be the candidate departing path to v avoiding edge on the subpath yz of \mathcal{P} . Then, there is a vertex $(v, |R|)$ in G . Moreover the shortest path from (s) to $(v, |R|)$ in the graph G is of length $|R|$, that is $|R| = |(s)(v, |R|)|_G$*

In Lemma 30, we will show that Property 29 is true for all the nodes added during initialization. Also, we will create $DEP(v)$ for each v during initialization. We will store candidate departing paths in $DEP(v)$ in increasing order of lengths. Given a departing path R , we will assume that we will store the following information about R in $DEP()$.

1. The endpoints of R .
2. The weight of path R .
3. The last edge of R and its weight.
4. $DP(R)$.

After initialization, we will run a variant of Dijkstra's algorithm in G . To this end, we will construct a min-heap H in which we will store all the departing paths that we have discovered at any point in the algorithm. We use the first two points of Definition 9 to select the minimum element from H , i.e., given two candidate departing paths R and R' , R is *smaller* than R' if $|R| < |R'|$ or $|R| = |R'|$ and $DEP(R)$ is closer to s than $DEP(R')$. If $DEP(R) = DEP(R')$, then we can break ties arbitrarily.

We now explain the adaptation of Dijkstra's algorithm. After initialization, for each $(v, |sv|)$, and for each neighbor w of v , we add the departing path $sv + (v, w)$ in H if $w \notin \mathcal{P}$. Then, we go over all the elements of the heap till it is empty. Let us assume that R is the minimum element of the heap and it ends at v and $(u, v) \in \widehat{G}$ is the last edge of R . This implies that R was added in H while processing a candidate departing path for u . Let this path be R_u . Thus, there is a node $(u, |R_u|)$ in G . We now need to decide whether R is a candidate departing path for v .

To this end, we look at the last candidate departing path added by us in $\text{DEP}(v)$, let it be Q . We then check if the $d_{\widehat{G}}(s, \text{DP}(R))$ is less than $d_{\widehat{G}}(s, \text{DP}(Q))$. If yes, then we have found a new candidate departing path to v that avoids all edges in $[\text{DP}(R), \text{DP}(Q)]$ of \mathcal{P} . Thus, we will add the vertex $(v, |R|)$ in the graph with an edge of weight $wt_{\widehat{G}}(u, v)$ from $(u, |R_u|)$. Also, for each neighbor w of v , we will add the departing path $R + (v, w)$ to the heap if $w \notin \mathcal{P}$. For the formal pseudocode, see the full version [13] of this paper.

5.5 Correctness and running time of the algorithm storing candidate departing paths

We claim that, the time taken to construct the $\text{DEP}()$ data-structure at a node of the binary tree with graph \widehat{G} is $O(m_{\widehat{G}}\sqrt{n})$. Moreover, the size of the data-structure is $O(n_{\widehat{G}}\sqrt{n})$. At first, we prove that our algorithm stores correct lengths of all candidate departing paths.

► **Lemma 30.** *Let R be a candidate departing path to v where $v \in \widehat{G} \setminus \mathcal{P}$. Let yz be the maximal subpath of \mathcal{P} such that R is the candidate departing path for edges in yz . Then $(v, |R|) \in G$ and satisfies Property 29.*

Proof. We will prove the above lemma using induction on the weighted distance of a vertex from (s) in G . During initialization, for each $v \in G \setminus \mathcal{P}$, we add a vertex $(v, |sv|_{\widehat{G}}) \in G$. Also, the weight of the edge from (s) to $(v, |sv|_{\widehat{G}})$ is $|sv|_{\widehat{G}}$. We claim that the statement of the lemma is true for the smallest candidate departing path to v . Indeed, using Lemma 22, $(sv)_{\widehat{G}}$ is the a replacement path for the edges in subpath pr on \mathcal{P} where $p = \text{LCA}_{\widehat{G}}(v, r)$. Also, $(sv)_{\widehat{G}}$ is the smallest replacement path because it is the shortest path from s to v in \widehat{G} . Thus, the base case is true for all $v \in \widehat{G} \setminus \mathcal{P}$.

Let us now assume that the statement of the lemma is true for all candidate departing paths to v with length $< |R|$. Let the second last vertex of R be u . Since R is a candidate departing path, even $R \setminus (u, v)$ is a candidate departing path. Since $R[s, u]$ has length less than R , by induction hypothesis, there is a vertex $(u, |R \setminus (u, v)|)$ in G . Also there is at least one replacement path in $\text{DEP}(v)$ of weight less than $|R|$ – as we have added $(sv)_{\widehat{G}}$ in $\text{DEP}(v)$. Let Q be a candidate departing path of largest weight less than the weight of R . Let us also assume that Q avoids edge on subpath $zz' \in \mathcal{P}$. Thus, using Lemma 26, $\text{DP}(Q) = z$. Since $|Q| < |R|$, using Lemma 27, $\text{DP}(R)$ lies above $\text{DP}(Q)$ on \mathcal{P} . Using the induction hypothesis, there is a vertex $(v, |Q|)$ in G .

We will now show that our algorithm will add $(v, |R|)$ in G . There are four cases:

1. Our algorithm does not add any vertex for v after $(v, |Q|)$
But our algorithm does process the vertex $(u, |R \setminus (u, v)|)$. Thus, it will check each neighbour of u . When it checks the neighbor v , it will add the departing path R in the heap H . Thus, we will add the vertex $(v, |R|)$ in G while processing R , leading to a contradiction.
2. Our algorithm adds a vertex $(v, |R'|)$ where $\text{DP}(R)$ lies above $\text{DP}(R')$ on \mathcal{P}
We claim that the weight of R' cannot be less than the weight of R as then R is not the candidate departing path for all the edges in $\text{DP}(R')z$ subpath, contradicting the statement of the lemma. So let us assume that $|R| = |R'|$. But then $\text{DP}(R)$ lies above $\text{DP}(R')$ on \mathcal{P} . Thus, the min-heap will give preference to the replacement path R first, and our algorithm will make the vertex $(v, |R|)$. Again a contradiction.
3. Our algorithm adds a vertex $(v, |R'|)$ where $\text{DP}(R)$ lies below $\text{DP}(R')$ on \mathcal{P}
Again, we claim that the weight of R' cannot be less than the weight of R as then it R is not the replacement path for all edges in yz subpath, contradicting the statement of the

lemma. So let us assume that $|R| = |R'|$. But $\text{DP}(R')$ is closer to s than $\text{DP}(R)$. Thus, R' should be the candidate departing path avoiding edges of yz . This contradicts our assumption that R is the candidate departing path for all edges in yz .

4. Our algorithm adds a vertex $(v, |R'|)$ where $\text{DP}(R) = \text{DP}(R')$
 $|R'|$ cannot be less than $|R|$ as otherwise our algorithm will give preference to path R .
 But, if $|R| = |R'|$, then there is a vertex $(v, |R'|) = (v, |R|)$ in G .

So, we add the node $(v, |R|)$ in the graph G . At that moment, we also add an edge from $(u, R \setminus (u, v))$ to $(v, |R|)$ in G with weight $wt_{\widehat{G}}(u, v)$. Using induction, $|(s)(u, R \setminus (u, v))|_G = |R \setminus (u, v)|$. Thus, $|(s)(v, |R|)|_G = |R \setminus (u, v)| + wt_{\widehat{G}}(u, v) = |R|$. This completes our proof. \blacktriangleleft

Let's determine the running time of our algorithm. Using Lemma 23, for each $v \in \widehat{G}$, we make $O(\sqrt{n})$ entries in $\text{DEP}(v)$. In other words, we make $O(\sqrt{n})$ nodes of type (v, \cdot) in G . Whenever we add a node $(v, |R|)$ in G , we see all the edges of v . This implies that the total time taken to process all the vertices of v in G is $O(\sqrt{n} \deg_{\widehat{G}}(v))$. Summing it over all the vertices gives us the bound of $O(m_{\widehat{G}}\sqrt{n})$. Using a similar calculation, the total size of our data-structure for \widehat{G} is $O(n_{\widehat{G}}\sqrt{n})$. Thus, we claim the following lemma:

► **Lemma 31.** *The time taken to construct the $\text{DEP}()$ data-structure at a node of the binary tree with graph \widehat{G} is $O(m_{\widehat{G}}\sqrt{n})$. Moreover, the size of the data-structure is $O(n_{\widehat{G}}\sqrt{n})$.*

5.6 Querying for a candidate departing path

In this section, we describe how to find a candidate departing path using our data-structure $\text{DEP}()$. Let $t \in \widehat{G} \setminus \mathcal{P}$ and $e \in \mathcal{P}$ be an edge on st path. Let $st \diamond e$ be a candidate departing path, then we can find it using the algorithm given in the full version of this paper [13].

In this algorithm, we perform a binary search in $\text{DEP}(t)$ to find two consecutive paths R and Q such that e lies in the interval $[\text{DEP}(R), \text{DEP}(Q)]$ of \mathcal{P} . Using Lemma 27 and Lemma 26, R is the candidate departing path avoiding e .

6 Construction time, size and query time of the SDO(1)

In this section, we show that the construction time of our algorithm is $\widetilde{O}(m\sqrt{n})$. We also bound the size of the data-structure of our algorithm by $\widetilde{O}(n\sqrt{n})$. We also design a query algorithm with a query time $\widetilde{O}(1)$. This proves the main result of the paper.

At the root of \mathcal{T} , except for the recursions, we claim that constructing all other data-structures take $O(m\sqrt{n})$ time. This is because, the construction time is dominated by the time to construct $\text{DEP}()$, which using Lemma 31, is $O(m\sqrt{n})$. At the second level of the tree \mathcal{T} , we have two nodes. In the left child, we have the graph $G_M \cup X$. This graph has $m_{G_M} + n_{G_M}$ edges and n_{G_M} vertices. Again applying Lemma 31, the time taken to construct all the data-structures in the left child of root is $(m_{G_M} + n_{G_M})\sqrt{n}$. In the right child of the root, we have the graph $G_N \cup Y$. This graph has $m_{G_N} + n_{G_N}$ edges and $n_{G_N} + 1$ vertices. The +1 is for the new root in G_N . Again applying Lemma 31, the time taken to construct all the data-structures in the right child of root is $(m_{G_N} + n_{G_N})\sqrt{n}$. Thus, the total time taken at the second level of \mathcal{T} is $(m_{G_M} + n_{G_M})\sqrt{n} + (m_{G_N} + n_{G_N})\sqrt{n}$. Since $m_{G_M} + m_{G_N} \leq m$ and $n_{G_M} + n_{G_N} = n + 1$, the total time taken is $\leq (m + n + 1)\sqrt{n}$. Note that $n_{G_M} + n_{G_N} = n + 1$ because r is shared both by G_M and G_N . Since, the number of nodes in \mathcal{T} is $O(n)$, we claim that the number of vertices shared by sibling graphs at any level of \mathcal{T} is $O(n)$. Similar to the second level, we claim that the time taken at level ℓ is $\widetilde{O}((m + n + \#\text{nodes shared at level } \ell)\sqrt{n}) = \widetilde{O}((m + n)\sqrt{n})$. We can assume that our graph

G is connected as we need not even process a component that is not reachable from our source s . Thus, the previous running time bound is equal to $\tilde{O}(m\sqrt{n})$. Since the height of the tree is $\tilde{O}(1)$, the total time taken to construct our data-structure is $\tilde{O}(m\sqrt{n})$. Using the same argument, we can bound the size of the data-structure of our algorithm by $\tilde{O}(n\sqrt{n})$.

6.1 The Query Algorithm

In this section, we will design our query algorithm that will take s, t, e as its parameter. Additionally, it also takes the root of the tree \mathcal{T} as a parameter which contains data structures of the main graph G . The algorithm then basically goes over all the possible cases described in Section 4 (Please see algorithm 3). Also, the algorithm compares that output with the best candidate departing path given by and return the minimum among them.

■ **Algorithm 1** QUERY(s, t, e, \mathcal{T}).

```

1 mindist  $\leftarrow$   $\infty$ ;
  /* Section 4.1 */
2 if  $e \in G_M, t \in G_N$  then
3   if  $e \in \mathcal{P}$  then
4     /* if  $st \diamond e$  happens to be departing */
4     mindist  $\leftarrow$  QUERY-DEP( $s, t, e$ ) using DEP() data-structure at  $\mathcal{T}$ ;
5     /* if  $st \diamond e$  happens to be jumping */
5     mindist  $\leftarrow$  min{mindist,  $|sr \diamond e| + |rt|$ }
6   else
7     mindist  $\leftarrow$   $|st|$ 
  /* Section 4.2 */
8 if  $e \in G_M, t \in G_M$  then
9   /* Section 4.2.1 */
9   if  $e \in \mathcal{P}$  then
10    /* if  $st \diamond e$  happens to be departing */
10    mindist  $\leftarrow$  QUERY-DEP( $s, t, e$ ) using DEP() data-structure at  $\mathcal{T}$ ;
11    /* if  $st \diamond e$  happens to be jumping */
11    mindist  $\leftarrow$  min(mindist,  $|sr \diamond e| + |rt|$ )
12   /* Section 4.2.2 */
12   if  $e \in G_M \setminus \mathcal{P}$  then
13     mindist  $\leftarrow$  QUERY( $s, t, e$ , left child of  $\mathcal{T}$ )
  /* Section 4.3 */
14 if  $e \in G_N, t \in G_M$  then
15   mindist  $\leftarrow$   $|st|$ ;
  /* Section 4.4 */
16 if  $e \in G_N, t \in G_N$ , then
17   mindist  $\leftarrow$  QUERY( $s, t, e$ , right child of  $\mathcal{T}$ );
  /* Section 4.5 */
18 if one endpoint of  $e$  is in  $G_M$  and other in  $G_N$  then
19   mindist  $\leftarrow$   $|st|$ ;
20 return mindist;
```

The reader can see that the time taken by the algorithm (excluding recursion) is $\tilde{O}(1)$. Since, at each step in this algorithm, we either go to the left child of a node in the tree \mathcal{T} or to the right child, the number of recursive steps in this algorithm is $\tilde{O}(1)$ or to be specific $O(\log n)$. Then for each child the call to $\text{QUERY-DEP}(s, t, e)$ takes $O(\log n)$ time. This implies that the running time of the query algorithm is $O(\log^2 n)$ or $\tilde{O}(1)$. Thus, we have proven the main theorem of the paper.

References

- 1 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, pages 88–94, 2000.
- 2 Aaron Bernstein. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '10*, pages 742–755, USA, 2010. Society for Industrial and Applied Mathematics.
- 3 Aaron Bernstein and David Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 101–110. ACM, 2009.
- 4 Aaron Bernstein and David R. Karger. Improved distance sensitivity oracles via random sampling. 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 34–43. SIAM, 2008. URL: <http://dl.acm.org/citation.cfm?id=1347082.1347087>.
- 5 Davide Bilò, Sarel Cohen, Tobias Friedrich, and Martin Schirneck. Near-optimal deterministic single-source distance sensitivity oracles. In *Accepted in Annual European Symposium on Algorithms, ESA 2021*, 2021.
- 6 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 18:1–18:14, 2016.
- 7 Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving distances in very faulty graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 73:1–73:14, 2017.
- 8 Shiri Chechik and Sarel Cohen. Near optimal algorithms for the single source replacement paths problem. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2090–2109, 2019.
- 9 Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. $(1 + \epsilon)$ -approximate f -sensitive distance oracles. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1479–1496, 2017.
- 10 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault tolerant spanners for general graphs. *SIAM J. Comput.*, 39(7):3403–3423, 2010. doi:10.1137/090758039.
- 11 Shiri Chechik and Ofer Magen. Near optimal algorithm for the directed single source replacement paths problem. 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 81:1–81:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 12 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

- 13 Dipan Dey and Manoj Gupta. Near optimal algorithm for fault tolerant distance oracle and single source replacement path problem. *CoRR*, abs/2206.15016, 2022. doi:10.48550/arXiv.2206.15016.
- 14 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 506–515, USA, 2009. Society for Industrial and Applied Mathematics.
- 15 Ran Duan and Seth Pettie. Approximating maximum weight matching in near-linear time. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 673–682, Washington, DC, USA, 2010. IEEE Computer Society. doi:10.1109/FOCS.2010.70.
- 16 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 748–757. IEEE, 2012.
- 17 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster replacement paths and distance sensitivity oracles. *ACM Trans. Algorithms*, 16(1), December 2019. doi:10.1145/3365835.
- 18 Manoj Gupta, Rahul Jain, and Nitiksha Modi. Multiple source replacement path problem. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 339–348. ACM, 2020.
- 19 Manoj Gupta and Shahbaz Khan. Multiple source dual fault tolerant BFS trees. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 127:1–127:15, 2017.
- 20 Manoj Gupta and Aditi Singh. Generic single edge fault tolerant exact distance oracle. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 72:1–72:15, 2018.
- 21 John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 252–259, 2001.
- 22 Kavindra Malik, Ashok K Mittal, and Santosh K Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, 1989.
- 23 Enrico Nardelli, Guido Proietti, and Peter Widmayer. Finding the most vital node of a shortest path. *Theor. Comput. Sci.*, 296(1):167–177, 2003.
- 24 Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1-2):166–196, 2001.
- 25 Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 481–490, 2015.
- 26 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 779–790, 2013.
- 27 Liam Roditty. On the k shortest simple paths problem in weighted directed graphs. *SIAM J. Comput.*, 39:2363–2376, January 2010. doi:10.1137/080730950.
- 28 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. In *Proceedings of the 32nd International Conference on Automata, Languages and Programming, ICALP'05*, pages 249–260, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11523468_21.
- 29 Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Trans. Algorithms*, 9(2), March 2013. doi:10.1145/2438645.2438646.

Fast Computation of Zigzag Persistence

Tamal K. Dey ✉

Department of Computer Science, Purdue University, West Lafayette, IN, USA

Tao Hou ✉

School of Computing, DePaul University, Chicago, IL, USA

Abstract

Zigzag persistence is a powerful extension of the standard persistence which allows deletions of simplices besides insertions. However, computing zigzag persistence usually takes considerably more time than the standard persistence. We propose an algorithm called FASTZIGZAG which narrows this efficiency gap. Our main result is that an input simplex-wise zigzag filtration can be converted to a *cell-wise non-zigzag* filtration of a Δ -complex with the same length, where the cells are copies of the input simplices. This conversion step in FASTZIGZAG incurs very little cost. Furthermore, the barcode of the original filtration can be easily read from the barcode of the new cell-wise filtration because the conversion embodies a series of *diamond switches* known in topological data analysis. This seemingly simple observation opens up the vast possibilities for improving the computation of zigzag persistence because any efficient algorithm/software for standard persistence can now be applied to computing zigzag persistence. Our experiment shows that this indeed achieves substantial performance gain over the existing state-of-the-art softwares.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Mathematics of computing \rightarrow Algebraic topology

Keywords and phrases zigzag persistence, persistent homology, fast computation

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.43

Supplementary Material *Software (Source Code)*: <https://github.com/taohou01/fzz>

Funding This research is partially supported by NSF grant CCF 2049010.

Acknowledgements We thank the Stanford Computer Graphics Laboratory and Ryan Holmes for providing the triangular meshes used in the experiment of this paper.

1 Introduction

Standard persistent homology defined over a growing sequence of simplicial complexes is a fundamental tool in topological data analysis (TDA). Since the advent of persistence algorithm [18] and its algebraic understanding [30], various extensions of the basic concept have been explored [6, 8, 12, 13]. Among these extensions, zigzag persistence introduced by Carlsson and de Silva [6] is an important one. It empowered TDA to deal with filtrations where both insertion and deletion of simplices are allowed. In practice, allowing deletion of simplices does make the topological tool more powerful. For example, in dynamic networks [15, 21] a sequence of graphs may not grow monotonically but can also shrink due to disappearance of vertex connections. Furthermore, zigzag persistence seems to be naturally connected with the computations involving multiparameter persistence, see e.g. [16, 17].

Zigzag persistence possesses some key differences from standard persistence. For example, unlike standard (non-zigzag) modules which decompose into only finite and infinite intervals, zigzag modules decompose into four types of intervals (see Definition 2). Existing algorithms for computing zigzag persistence from a zigzag filtration [8, 22, 23, 24] are all based on maintaining explicitly or implicitly a consistent basis throughout the filtration. This makes these algorithms for zigzag persistence more involved and hence slower in practice than



© Tamal K. Dey and Tao Hou;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 43; pp. 43:1–43:15
Leibniz International Proceedings in Informatics



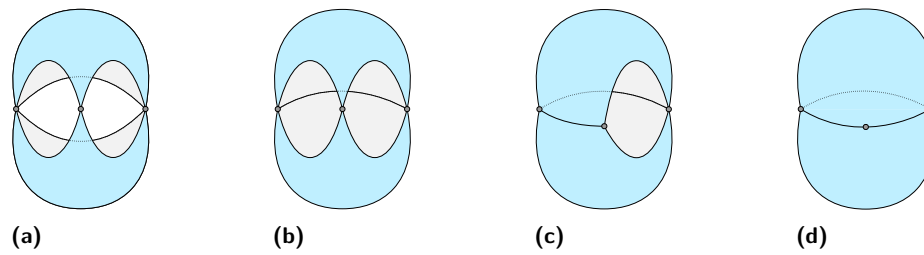
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

algorithms for the non-zigzag version though they have the same time complexity [25]. We sidestep the bottleneck of maintaining an explicit basis and propose an algorithm called FASTZIGZAG, which converts the input zigzag filtration to a *non-zigzag* filtration with an efficient strategy for mapping barcodes of the two bijectively. Then, we can apply any efficient algorithm for standard persistence on the resulting non-zigzag filtration to compute the barcode of the original filtration. Considering the abundance of optimizations [2, 3, 4, 5, 10, 11] of standard persistence algorithms and a recent GPU acceleration [29], the conversion in FASTZIGZAG enables zigzag persistence computation to take advantage of any existing or future improvements on standard persistence computation. Our implementation, which uses the PHAT [4] software for computing standard persistence, shows substantial performance gain over existing state-of-the-art softwares [26, 28] for computing zigzag persistence (see Section 3.5). We make our software publicly available through: <https://github.com/taohou01/fzz>.

To elaborate on the strategy of FASTZIGZAG, we first observe a special type of zigzag filtrations called *non-repetitive* zigzag filtrations in which a simplex (or more generally, a *cell*) is never added again once deleted. Such a filtration admits an *up-down* filtration as its canonical form that can be obtained by a series of *diamond switches* [6, 7, 8]. The up-down filtration can be further converted into a *non-zigzag* filtration again using diamond switches as in the Mayer-Vietoris pyramid presented in [8]. Individual switches are atomic tools that help us to show equivalence of barcodes, but we do not need to actually execute them in computation. Instead, we go straight to the final form of the filtration quite easily and efficiently. Finally, we observe that any zigzag filtration can be treated as a non-repetitive *cell-wise* filtration of a Δ -complex [20] consisting of multisets of input simplices. This means that each repeatedly added simplex is treated as a different cell in the Δ -complex, so that we can apply our findings for non-repetitive filtrations to arbitrary filtrations. The conversions described above are detailed in Section 3.

1.1 Related works

Zigzag persistence is essentially an A_n -type quiver [14] in mathematics which is first introduced to the TDA community by Carlsson and de Silva [6]. In their paper [6], Carlsson and de Silva also study the Mayer-Vietoris diamond used in this paper and propose an algorithm for computing zigzag barcodes from zigzag modules (i.e., an input is a sequence of vector spaces connected by linear maps encoded as matrices). Carlsson et al. [8] then propose an $O(mn^2)$ algorithm for computing zigzag barcodes from zigzag filtrations using a structure called right filtration. In their paper [8], Carlsson et al. also extend the classical sublevelset filtrations for functions on topological spaces by proposing levelset zigzag filtrations and show the equivalence of levelset zigzag with the extended persistence proposed by Cohen-Steiner et al. [12]. Maria and Outdot [22, 23] propose an alternative algorithm for computing zigzag barcodes by attaching a reversed standard filtration to the end of the partial zigzag filtration being scanned. Their algorithm maintains the barcode over the Surjective and Transposition Diamond on the constructed zigzag filtration [22, 23]. Maria and Schreiber [24] propose a Morse reduction preprocessing for zigzag filtrations which speeds up the zigzag barcode computation. Carlsson et al. [9] discuss some matrix factorization techniques for computing zigzag barcodes from zigzag modules, which, combined with a divide-and-conquer strategy, lead to a parallel algorithm for computing zigzag persistence. Almost all algorithms reviewed so far have a cubic time complexity. Milosavljević et al. [25] establish an $O(m^\omega)$ theoretical complexity for computing zigzag persistence from filtrations, where $\omega < 2.37286$ is the matrix multiplication exponent [1]. Recently, Dey and Hou [15] propose near-linear algorithms for computing zigzag persistence from the special cases of graph filtrations, with the help of representatives defined for the intervals and some dynamic graph data structures.



■ **Figure 1** Examples of Δ -complexes with two triangles sharing 0, 1, 2, or 3 edges on their boundaries.

2 Preliminaries

Δ -complex. In this paper, we build filtrations on Δ -complexes which are extensions of simplicial complexes described in Hatcher [20]. These Δ -complexes are derived from a set of standard simplices by identifying the boundary of each simplex with other simplices while preserving the vertex orders. For distinction, building blocks of Δ -complexes (i.e., standard simplices) are called *cells*. Motivated by a construction from the input simplicial complex described in Algorithm 3.1, we use a *more restricted* version of Δ -complexes, where boundary cells of each p -cell are identified with *distinct* $(p - 1)$ -cells. Notice that this makes each p -cell combinatorially equivalent to a p -simplex. Hence, the difference of the Δ -complexes in this paper from the standard simplicial complexes is that common faces of two cells in the Δ -complexes can have more relaxed forms. For example, in Figure 1, two “triangles” (2-cells) in a Δ -complex having the same set of vertices can either share 0, 1, 2, or 3 edges in their boundaries; note that the two triangles in Figure 1d form a 2-cycle.

Formally, we define Δ -complexes recursively similar to the classical definition of *CW-complexes* [20] though it need not be as general; see Hatcher’s book [20] for a more general definition. Note that simplicial complexes are trivially Δ -complexes and therefore most definitions in this section target Δ -complexes.

► **Definition 1.** A Δ -complex is defined recursively with dimension:

1. A 0-dimensional Δ -complex K^0 is a set of points, each called a 0-cell.
2. A p -dimensional Δ -complex K^p , $p \geq 1$, is a quotient space of a $(p - 1)$ -dimensional Δ -complex K^{p-1} along with several standard p -simplices. The quotienting is realized by an attaching map $h : \partial(\sigma) \rightarrow K^{p-1}$ which identifies the boundary $\partial(\sigma)$ of each p -simplex σ with points in K^{p-1} so that h is a homeomorphism onto its image. We term the standard p -simplex σ with boundary identified to K^{p-1} as a *p -cell* in K^p . Furthermore, we have that the restriction of h to each proper face of σ is a homeomorphism onto a cell in K^{p-1} .

Notice that the original (more general) Δ -complexes [20] require specifying vertex orders when identifying the cells. However, the restricted Δ -complexes defined above do not require specifying such orders because we always identify the boundaries of cells by *homeomorphisms* and hence the vertex orders for identification are implicitly derived from a vertex order of a seeding cell.

Homology. Homology in this paper is defined on Δ -complexes, which is defined similarly as for simplicial complexes [20]. All homology groups are taken with \mathbb{Z}_2 -coefficients and therefore vector spaces mentioned in this paper are also over \mathbb{Z}_2 .

Zigzag filtration and barcode. A *zigzag filtration* (or simply *filtration*) is a sequence of Δ -complexes

$$\mathcal{F} : K_0 \leftrightarrow K_1 \leftrightarrow \cdots \leftrightarrow K_m,$$

in which each $K_i \leftrightarrow K_{i+1}$ is either a forward inclusion $K_i \hookrightarrow K_{i+1}$ or a backward inclusion $K_i \leftarrow K_{i+1}$. For computational purposes, we only consider *cell-wise* filtrations in this paper, i.e., each inclusion $K_i \leftrightarrow K_{i+1}$ is an addition or deletion of a *single* cell; such an inclusion is sometimes denoted as $K_i \xrightarrow{\sigma} K_{i+1}$ with σ indicating the cell being added or deleted.

We call \mathcal{F} as *non-repetitive* if whenever a cell σ is deleted from \mathcal{F} , the cell σ is never added again. We call \mathcal{F} an *up-down* filtration [8] if \mathcal{F} can be separated into two parts such that the first part contains only forward inclusions and the second part contains only backward ones, i.e., \mathcal{F} is of the form $\mathcal{F} : K_0 \hookrightarrow K_1 \hookrightarrow \cdots \hookrightarrow K_\ell \leftarrow K_{\ell+1} \leftarrow \cdots \leftarrow K_m$. Usually in this paper, filtrations start and end with empty complexes, e.g., $K_0 = K_m = \emptyset$ in \mathcal{F} .

Applying the p -th homology functor on \mathcal{F} induces a *zigzag module*:

$$H_p(\mathcal{F}) : H_p(K_0) \leftrightarrow H_p(K_1) \leftrightarrow \cdots \leftrightarrow H_p(K_m),$$

in which each $H_p(K_i) \leftrightarrow H_p(K_{i+1})$ is a linear map induced by inclusion. It is known [6, 19] that $H_p(\mathcal{F})$ has a decomposition of the form $H_p(\mathcal{F}) \simeq \bigoplus_{k \in \Lambda} \mathcal{I}^{[b_k, d_k]}$, in which each $\mathcal{I}^{[b_k, d_k]}$ is a special type of zigzag module called *interval module* over the interval $[b_k, d_k]$. The (multi-)set of intervals denoted as $\text{Pers}_p(\mathcal{F}) := \{[b_k, d_k] \mid k \in \Lambda\}$ is an invariant of \mathcal{F} and is called the *p -th barcode* of \mathcal{F} . Each interval in $\text{Pers}_p(\mathcal{F})$ is called a *p -th persistence interval* and is also said to be in dimension p . Frequently in this paper, we consider the barcode of \mathcal{F} in all dimensions $\text{Pers}_*(\mathcal{F}) := \bigsqcup_{p \geq 0} \text{Pers}_p(\mathcal{F})$.

► **Definition 2** (Open and closed birth/death). For a zigzag filtration $\mathcal{F} : \emptyset = K_0 \leftrightarrow K_1 \leftrightarrow \cdots \leftrightarrow K_m = \emptyset$, the start of any interval in $\text{Pers}_*(\mathcal{F})$ is called a **birth index** in \mathcal{F} and the end of any interval is called a **death index**. Moreover, a birth index b is said to be **closed** if $K_{b-1} \hookrightarrow K_b$ is a forward inclusion; otherwise, b is **open**. Symmetrically, a death index d is said to be **closed** if $K_d \leftarrow K_{d+1}$ is a backward inclusion; otherwise, d is **open**. The types of the birth/death ends classify intervals in $\text{Pers}_*(\mathcal{F})$ into four types: **closed-closed**, **closed-open**, **open-closed**, and **open-open**.

► **Remark 3.** If \mathcal{F} is a levelset zigzag filtration [8], then the open and closed ends defined above are the same as for levelset zigzag.

► **Remark 4.** An inclusion $K_i \leftrightarrow K_{i+1}$ in a cell-wise filtration either provides $i + 1$ as a birth index or provides i as a death index (but cannot provide both).

Mayer-Vietoris diamond. The algorithm in this paper draws upon the Mayer-Vietoris diamond proposed by Carlsson and de Silva [6] (see also [7, 8]), which relates barcodes of two filtrations differing by a local change:

► **Definition 5** (Mayer-Vietoris diamond [6]). Two cell-wise filtrations \mathcal{F} and \mathcal{F}' are related by a **Mayer-Vietoris diamond** if they are of the following forms (where $\sigma \neq \tau$):

$$\begin{array}{c} \mathcal{F} : \\ K_0 \longleftrightarrow \cdots \longleftrightarrow K_{j-1} \begin{array}{c} \nearrow \sigma \\ \searrow \tau \end{array} K_j \begin{array}{c} \nwarrow \tau \\ \nearrow \sigma \end{array} K_{j+1} \longleftrightarrow \cdots \longleftrightarrow K_m \\ \mathcal{F}' : \\ \phantom{K_{j-1}} \phantom{K_{j+1}} \\ \phantom{K_{j-1}} \phantom{K_{j+1}} \end{array} \quad (1)$$

In the above diagram, \mathcal{F} and \mathcal{F}' differ only in the complexes at index j and \mathcal{F}' is derived from \mathcal{F} by switching the addition of σ and deletion of τ . We also say that \mathcal{F}' is derived from \mathcal{F} by an **outward** switch and \mathcal{F} is derived from \mathcal{F}' by an **inward** switch.

► **Remark 6.** In Equation (1), we only provide a specific form of Mayer-Vietoris diamond which is sufficient for our purposes; see [6, 8] for a more general form. According to [6], the diamond in Equation (1) is a Mayer-Vietoris diamond because $K_j = K_{j-1} \cup K_{j+1}$ and $K'_j = K_{j-1} \cap K_{j+1}$.

We then have the following fact:

► **Theorem 7** (Diamond Principle [6]). *Given two cell-wise filtrations $\mathcal{F}, \mathcal{F}'$ related by a Mayer-Vietoris diamond as in Equation (1), there is a bijection from $\text{Pers}_*(\mathcal{F})$ to $\text{Pers}_*(\mathcal{F}')$ as follows:*

$\text{Pers}_*(\mathcal{F})$	$\text{Pers}_*(\mathcal{F}')$
$[b, j-1]; b \leq j-1$	$[b, j]$
$[b, j]; b \leq j-1$	$[b, j-1]$
$[j, d]; d \geq j+1$	$[j+1, d]$
$[j+1, d]; d \geq j+1$	$[j, d]$
$[j, j]$ of dimension p	$[j, j]$ of dimension $p-1$
$[b, d];$ all other cases	$[b, d]$

Note that the bijection preserves the dimension of the intervals except for $[j, j]$.

► **Remark 8.** In the above bijection, only an interval containing *some but not all* of $\{j-1, j, j+1\}$ maps to a different interval or different dimension.

3 FASTZIGZAG algorithm

In this section, we show that computing barcodes for an arbitrary zigzag filtration of simplicial complexes can be reduced to computing barcodes for a certain *non-zigzag* filtration of Δ -complexes. The resulting algorithm called FASTZIGZAG is more efficient considering that standard (non-zigzag) persistence admits faster algorithms [2, 3, 4, 5, 10, 11, 29] in practice. We confirm the efficiency with experiments in Section 3.5.

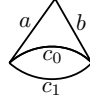
3.1 Overview

Given a *simplex-wise* zigzag filtration

$$\mathcal{F} : \emptyset = K_0 \xleftarrow{\sigma_0} K_1 \xleftarrow{\sigma_1} \dots \xleftarrow{\sigma_{m-1}} K_m = \emptyset$$

of simplicial complexes as input, the FASTZIGZAG algorithm has the following main procedure:

1. Convert \mathcal{F} into a *non-repetitive* zigzag filtration of Δ -complexes.
2. Convert the non-repetitive filtration to an *up-down* filtration.
3. Convert the up-down filtration to a *non-zigzag* filtration with the help of an *extended persistence* filtration.
4. Compute the standard persistence barcode, which is then converted to the barcode for the input filtration based on rules given in Proposition 15 and 19.



■ **Figure 2** The Δ -complex resulting from performing an inward switch around \hat{K}_4 for the example shown in Figure 5.

Step 1 is achieved by simply treating each repeatedly added simplex in \mathcal{F} as a new cell in the converted filtration (see also [25]). Throughout the section, we denote the converted non-repetitive, *cell-wise* filtration as

$$\hat{\mathcal{F}} : \emptyset = \hat{K}_0 \xleftarrow{\hat{\sigma}_0} \hat{K}_1 \xleftarrow{\hat{\sigma}_1} \dots \xleftarrow{\hat{\sigma}_{m-1}} \hat{K}_m = \emptyset.$$

Notice that each \hat{K}_i in $\hat{\mathcal{F}}$ is homeomorphic to K_i in \mathcal{F} , and hence $\text{Pers}_*(\mathcal{F}) = \text{Pers}_*(\hat{\mathcal{F}})$. However, we get an important difference between \mathcal{F} and $\hat{\mathcal{F}}$ by treating the simplicial complexes as Δ -complexes. For example, in Figure 5 presented later in this section, the first addition of edge c in \mathcal{F} corresponds to a cell c_0 in $\hat{\mathcal{F}}$ and its second addition in \mathcal{F} corresponds to a cell c_1 . Performing an inward switch around \hat{K}_4 (switching $\xleftarrow{c_0}$ and $\xleftarrow{c_1}$) turns \hat{K}_4 into a Δ -complex as shown in Figure 2. However, we cannot perform such a switch in \mathcal{F} which consists of simplicial complexes, because diamond switches require the switched simplices or cells to be different (see Definition 5).

In Section 3.2 and 3.3, we provide details for Step 2 and 3 as well as propositions for converting barcodes mentioned in Step 4. We summarize the filtration converting process in Section 3.4 by providing pseudocodes (Algorithm 3.1) and examples (Figure 5 and 6).

3.2 Conversion to up-down filtration

► **Proposition 9.** *For the filtration $\hat{\mathcal{F}}$, there is a cell-wise up-down filtration*

$$U : \emptyset = L_0 \hookrightarrow L_1 \hookrightarrow \dots \hookrightarrow L_n \hookleftarrow L_{n+1} \hookleftarrow \dots \hookleftarrow L_{2n} = \emptyset$$

derived from $\hat{\mathcal{F}}$ by a sequence of inward switches. Note that $m = 2n$

Proof. Let $\hat{K}_i \xleftarrow{\hat{\sigma}_i} \hat{K}_{i+1}$ be the first deletion in $\hat{\mathcal{F}}$ and $\hat{K}_j \xleftarrow{\hat{\sigma}_j} \hat{K}_{j+1}$ be the first addition after that. That is, $\hat{\mathcal{F}}$ is of the form

$$\hat{\mathcal{F}} : \hat{K}_0 \hookrightarrow \dots \hookrightarrow \hat{K}_i \xleftarrow{\hat{\sigma}_i} \hat{K}_{i+1} \xleftarrow{\hat{\sigma}_{i+1}} \dots \xleftarrow{\hat{\sigma}_{j-2}} \hat{K}_{j-1} \xleftarrow{\hat{\sigma}_{j-1}} \hat{K}_j \xleftarrow{\hat{\sigma}_j} \hat{K}_{j+1} \hookrightarrow \dots \hookrightarrow \hat{K}_m.$$

Since $\hat{\mathcal{F}}$ is non-repetitive, we have $\hat{\sigma}_{j-1} \neq \hat{\sigma}_j$. So we can switch $\xleftarrow{\hat{\sigma}_{j-1}}$ and $\xleftarrow{\hat{\sigma}_j}$ (which is an inward switch) to derive a filtration

$$\hat{K}_0 \hookrightarrow \dots \hookrightarrow \hat{K}_i \xleftarrow{\hat{\sigma}_i} \hat{K}_{i+1} \xleftarrow{\hat{\sigma}_{i+1}} \dots \xleftarrow{\hat{\sigma}_{j-2}} \hat{K}_{j-1} \xleftarrow{\hat{\sigma}_j} \hat{K}'_j \xleftarrow{\hat{\sigma}_{j-1}} \hat{K}_{j+1} \hookrightarrow \dots \hookrightarrow \hat{K}_m.$$

We then continue performing such inward switches (e.g., the next switch is on $\xleftarrow{\hat{\sigma}_{j-2}}$ and $\xleftarrow{\hat{\sigma}_j}$) to derive a filtration

$$\hat{\mathcal{F}}' : \hat{K}_0 \hookrightarrow \dots \hookrightarrow \hat{K}_i \xleftarrow{\hat{\sigma}_j} \hat{K}'_{i+1} \xleftarrow{\hat{\sigma}_i} \dots \xleftarrow{\hat{\sigma}_{j-3}} \hat{K}'_{j-1} \xleftarrow{\hat{\sigma}_{j-2}} \hat{K}'_j \xleftarrow{\hat{\sigma}_{j-1}} \hat{K}_{j+1} \hookrightarrow \dots \hookrightarrow \hat{K}_m.$$

Note that from $\hat{\mathcal{F}}$ to $\hat{\mathcal{F}}'$, the up-down “prefix” grows longer. We can repeat the above operations on the newly derived $\hat{\mathcal{F}}'$ until the entire filtration turns into an up-down one. ◀

Throughout the section, let

$$\mathcal{U} : \emptyset = L_0 \xleftarrow{\tau_0} \dots \xleftarrow{\tau_{n-1}} L_n \xleftarrow{\tau_n} \dots \xleftarrow{\tau_{2n-1}} L_{2n} = \emptyset$$

be the up-down filtration for $\hat{\mathcal{F}}$ as described in Proposition 9, where $m = 2n$. We also let $\hat{K} = L_n$.

In a cell-wise filtration, for a cell σ , let its addition (insertion) be denoted as $\downarrow\sigma$ and its deletion (removal) be denoted as $\uparrow\sigma$. From the proof of Proposition 9, we observe the following: during the transition from $\hat{\mathcal{F}}$ to \mathcal{U} , for any two additions $\downarrow\sigma$ and $\downarrow\sigma'$ in $\hat{\mathcal{F}}$ (and similarly for deletions), if $\downarrow\sigma$ is before $\downarrow\sigma'$ in $\hat{\mathcal{F}}$, then $\downarrow\sigma$ is also before $\downarrow\sigma'$ in \mathcal{U} . We then have the following fact:

► **Fact 10.** *Given the filtration $\hat{\mathcal{F}}$, to derive \mathcal{U} , one only needs to scan $\hat{\mathcal{F}}$ and list all the additions first and then the deletions, following the order in $\hat{\mathcal{F}}$.*

► **Remark 11.** Figure 3 gives an example of $\hat{\mathcal{F}}$ and its corresponding \mathcal{U} , where the additions and deletions in $\hat{\mathcal{F}}$ and \mathcal{U} follow the same order.

► **Definition 12** (Creator and destroyer). *For any interval $[b, d] \in \text{Pers}_*(\hat{\mathcal{F}})$, if $\hat{K}_{b-1} \xleftarrow{\hat{\sigma}_{b-1}} \hat{K}_b$ is forward (resp. backward), we call $\downarrow\hat{\sigma}_{b-1}$ (resp. $\uparrow\hat{\sigma}_{b-1}$) the **creator** of $[b, d]$. Similarly, if $\hat{K}_d \xleftarrow{\hat{\sigma}_d} \hat{K}_{d+1}$ is forward (resp. backward), we call $\downarrow\hat{\sigma}_d$ (resp. $\uparrow\hat{\sigma}_d$) the **destroyer** of $[b, d]$.*

By inspecting the interval mapping in the Diamond Principle, we have the following fact:

► **Proposition 13.** *For two cell-wise filtrations $\mathcal{L}, \mathcal{L}'$ related by a Mayer-Vietoris diamond, any two intervals of $\text{Pers}_*(\mathcal{L})$ and $\text{Pers}_*(\mathcal{L}')$ mapped by the Diamond Principle have the same set of creator and destroyer, though the creator and destroyer may swap. This observation combined with Proposition 9 implies that there is a bijection from $\text{Pers}_*(\mathcal{U})$ to $\text{Pers}_*(\hat{\mathcal{F}})$ s.t. every two corresponding intervals have the same set of creator and destroyer.*

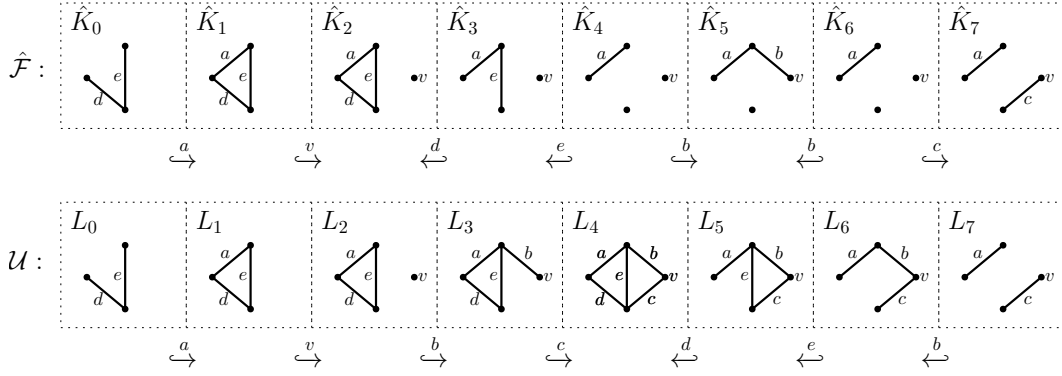
► **Remark 14.** The only time when the creator and destroyer swap in a Mayer-Vietoris diamond is when the interval $[j, j]$ for the upper filtration in Equation (1) turns into the same interval (of one dimension lower) for the lower filtration.

Consider the example in Figure 3 for an illustration of Proposition 13. In the example, $[1, 2] \in \text{Pers}_1(\hat{\mathcal{F}})$ corresponds to $[1, 4] \in \text{Pers}_1(\mathcal{U})$, where their creator is $\downarrow a$ and their destroyer is $\uparrow d$. Moreover, $[4, 6] \in \text{Pers}_0(\hat{\mathcal{F}})$ corresponds to $[4, 5] \in \text{Pers}_1(\mathcal{U})$. The creator of $[4, 6] \in \text{Pers}_0(\hat{\mathcal{F}})$ is $\uparrow e$ and the destroyer is $\downarrow c$. Meanwhile, $[4, 5] \in \text{Pers}_1(\mathcal{U})$ has the same set of creator and destroyer but the roles swap.

For any $\downarrow\sigma$ or $\uparrow\sigma$ in $\hat{\mathcal{F}}$, let $\text{id}_{\hat{\mathcal{F}}}(\downarrow\sigma)$ or $\text{id}_{\hat{\mathcal{F}}}(\uparrow\sigma)$ denote the index (position) of the addition or deletion. For example, for an addition $\hat{K}_i \xleftarrow{\hat{\sigma}_i} \hat{K}_{i+1}$ in $\hat{\mathcal{F}}$, $\text{id}_{\hat{\mathcal{F}}}(\downarrow\hat{\sigma}_i) = i$. Proposition 13 indicates the following explicit mapping from $\text{Pers}_*(\mathcal{U})$ to $\text{Pers}_*(\hat{\mathcal{F}})$:

► **Proposition 15.** *There is a bijection from $\text{Pers}_*(\mathcal{U})$ to $\text{Pers}_*(\hat{\mathcal{F}})$ which maps each $[b, d] \in \text{Pers}_p(\mathcal{U})$ by the following rule:*

Type	Condition		Type	Interval in $\text{Pers}_*(\hat{\mathcal{F}})$	Dim
closed-open	-	\mapsto	closed-open	$[\text{id}_{\hat{\mathcal{F}}}(\downarrow\tau_{b-1}) + 1, \text{id}_{\hat{\mathcal{F}}}(\downarrow\tau_d)]$	p
open-closed	-	\mapsto	open-closed	$[\text{id}_{\hat{\mathcal{F}}}(\uparrow\tau_{b-1}) + 1, \text{id}_{\hat{\mathcal{F}}}(\uparrow\tau_d)]$	p
closed-closed	$\text{id}_{\hat{\mathcal{F}}}(\downarrow\tau_{b-1}) < \text{id}_{\hat{\mathcal{F}}}(\uparrow\tau_d)$	\mapsto	closed-closed	$[\text{id}_{\hat{\mathcal{F}}}(\downarrow\tau_{b-1}) + 1, \text{id}_{\hat{\mathcal{F}}}(\uparrow\tau_d)]$	p
	$\text{id}_{\hat{\mathcal{F}}}(\downarrow\tau_{b-1}) > \text{id}_{\hat{\mathcal{F}}}(\uparrow\tau_d)$	\mapsto	open-open	$[\text{id}_{\hat{\mathcal{F}}}(\uparrow\tau_d) + 1, \text{id}_{\hat{\mathcal{F}}}(\downarrow\tau_{b-1})]$	$p-1$



■ **Figure 3** An example of filtration $\hat{\mathcal{F}}$ and its corresponding up-down filtration \mathcal{U} . For brevity, $\hat{\mathcal{F}}$ does not start and end with empty complexes (which can be treated as a truncated case).

► **Remark 16.** Notice that $\text{Pers}_*(\mathcal{U})$ contains no open-open intervals. However, a closed-closed interval $[b, d] \in \text{Pers}_p(\mathcal{U})$ turns into an open-open interval in $\text{Pers}_{p-1}(\hat{\mathcal{F}})$ when $\text{id}_{\hat{\mathcal{F}}}(\downarrow\tau_{b-1}) > \text{id}_{\hat{\mathcal{F}}}(\uparrow\tau_d)$. Such a change happens when a closed-closed interval turns into a single point interval $[j, j]$ during the sequence of outward switches, after which the closed-closed interval $[j, j]$ becomes an open-open interval $[j, j]$ with a dimension shift (see Theorem 7).

► **Remark 17.** Although it may take $O(m^2)$ diamond switches to go from $\hat{\mathcal{F}}$ to \mathcal{U} or from \mathcal{U} to $\hat{\mathcal{F}}$ as indicated in Proposition 9, we observe that these switches do not need to be actually executed in the algorithm. To convert the intervals in $\text{Pers}_*(\mathcal{U})$ to those in $\text{Pers}_*(\hat{\mathcal{F}})$, we only need to follow the mapping in Proposition 15, which takes constant time per interval.

We can take the example in Figure 3 for the mapping in Proposition 15. The interval $[4, 5] \in \text{Pers}_1(\mathcal{U})$ is a closed-closed one whose creator is $\downarrow c$ and destroyer is $\uparrow e$. We have that $\text{id}_{\hat{\mathcal{F}}}(\downarrow c) = 6 > \text{id}_{\hat{\mathcal{F}}}(\uparrow e) = 3$. So the corresponding interval in $\text{Pers}_0(\hat{\mathcal{F}})$ is

$$[\text{id}_{\hat{\mathcal{F}}}(\uparrow e) + 1, \text{id}_{\hat{\mathcal{F}}}(\downarrow c)] = [4, 6].$$

3.3 Conversion to non-zigzag filtration

We first convert the up-down filtration \mathcal{U} to an extended persistence [12] filtration \mathcal{E} , which is then easily converted to an (absolute) non-zigzag filtration using the “coning” technique [12].

Inspired by the Mayer-Vietoris pyramid in [8], we relate $\text{Pers}_*(\mathcal{U})$ to the barcode of the filtration \mathcal{E} defined as:

$$\mathcal{E} : \emptyset = L_0 \hookrightarrow \cdots \hookrightarrow L_n = (\hat{K}, L_{2n}) \hookrightarrow (\hat{K}, L_{2n-1}) \hookrightarrow \cdots \hookrightarrow (\hat{K}, L_n) = (\hat{K}, \hat{K})$$

where $L_n = \hat{K} = (\hat{K}, L_{2n} = \emptyset)$. When denoting the persistence intervals of \mathcal{E} , we let the increasing index for the first half of \mathcal{E} continue to the second half, i.e., (\hat{K}, L_{2n-1}) has index $n+1$ and (\hat{K}, L_n) has index $2n$. Then, it can be verified that an interval $[b, d] \in \text{Pers}_*(\mathcal{E})$ for $b < n < d$ starts with the complex L_b and ends with (\hat{K}, L_{3n-d}) .

► **Remark 18.** A filtration in extended persistence [12] is originally defined for a PL function f , where the first half is the lower-star filtration of f and the second half (the relative part) is derived from the upper-star filtration of f . The filtration \mathcal{E} defined above is a generalization of the one in [12].

► **Proposition 19.** *There is a bijection from $\text{Pers}_*(\mathcal{E})$ to $\text{Pers}_*(\mathcal{U})$ which maps each $[b, d] \in \text{Pers}_*(\mathcal{E})$ of dimension p by the following rule:*

Type	Condition		Type	Interv. in $\text{Pers}_*(\mathcal{U})$	Dim
Ord	$d < n$	\mapsto	closed-open	$[b, d]$	p
Rel	$b > n$	\mapsto	open-closed	$[3n - d, 3n - b]$	$p-1$
Ext	$b \leq n \leq d$	\mapsto	closed-closed	$[b, 3n - d - 1]$	p

► **Remark 20.** The types “Ord”, “Rel”, and “Ext” for intervals in $\text{Pers}_*(\mathcal{E})$ are as defined in [12], which stand for intervals from the *ordinary* sub-barcode, the *relative* sub-barcode, and the *extended* sub-barcode.

► **Remark 21.** The above proposition can also be stated by associating the creators and destroyers as in Proposition 13 and 15. The association of additions in the first half of \mathcal{U} and \mathcal{E} is straightforward and the deletion of a σ in \mathcal{U} is associated with the addition of σ (to the second complex in the pair) in \mathcal{E} . Then, corresponding intervals in $\text{Pers}_*(\mathcal{E})$ and $\text{Pers}_*(\mathcal{U})$ in the above proposition also have the same set of creators and destroyers. Combined with Proposition 13, we further have that intervals in $\text{Pers}_*(\mathcal{F})$ and $\text{Pers}_*(\mathcal{E})$ can be associated by a bijection where corresponding intervals have the same pairs of simplices though they may switch roles of being creators and destroyers.

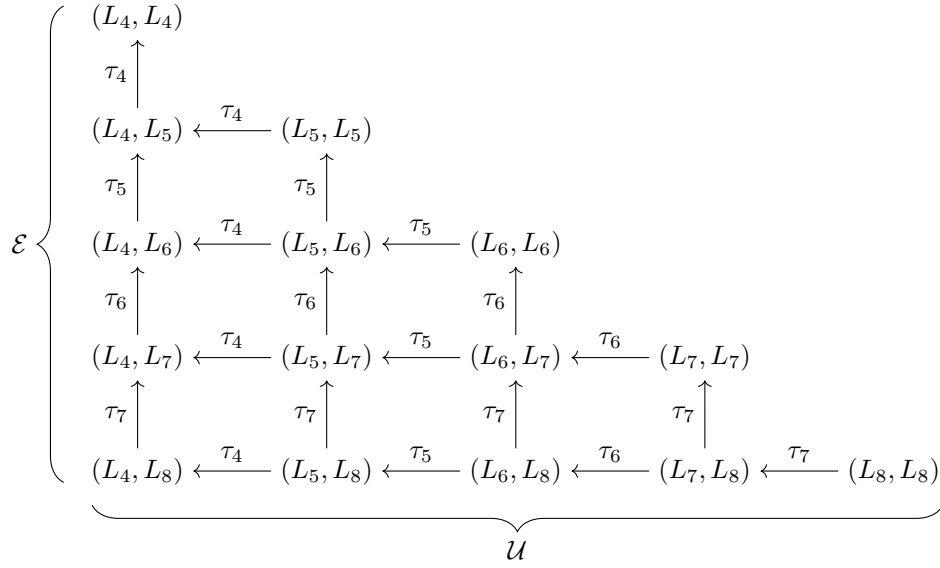
Proof. We can build a Mayer-Vietoris pyramid relating the second half of \mathcal{E} and the second half of \mathcal{U} similar to the one in [8]. A pyramid for $n = 4$ is shown in Figure 4, where the second half of \mathcal{E} is along the left side of the triangle and the second half of \mathcal{U} is along the bottom. In Figure 4, we represent the second half of \mathcal{E} and \mathcal{U} in a slightly different way considering that $L_4 = \hat{K}$ and $L_8 = \emptyset$. Also, each vertical arrow indicates the addition of a simplex in the second complex of the pair and each horizontal arrow indicates the deletion of a simplex in the first complex.

To see the correctness of the mapping, we first note that each square in the pyramid is a (more general version of) Mayer-Vietoris diamond as defined in [8]. Then, the mapping stated in the proposition can be verified using the Diamond Principle (Theorem 7). However, there is a quicker way to verify the mapping by observing the following: corresponding intervals in $\text{Pers}_*(\mathcal{E})$ and $\text{Pers}_*(\mathcal{U})$ have the same set of creator and destroyer if we ignore whether it is the addition or deletion of a simplex. For example, an interval in $\text{Pers}_*(\mathcal{E})$ may be created by the addition of a simplex σ in the first half of \mathcal{E} and destroyed by the addition of another simplex σ' in the second half of \mathcal{E} . Then, its corresponding interval in $\text{Pers}_*(\mathcal{U})$ is also created by the addition of σ in the first half but destroyed by the *deletion* of σ' in the second half. Note that the dimension change for the case $b > n$ is caused by the swap of creator and destroyer. ◀

By Proposition 15 and 19, we only need to compute $\text{Pers}_*(\mathcal{E})$ in order to compute $\text{Pers}_*(\mathcal{F})$. The barcode of \mathcal{E} can be computed using the “coning” technique [12], which converts \mathcal{E} into an (absolute) non-zigzag filtration $\hat{\mathcal{E}}$. Specifically, let ω be a vertex different from all vertices in \hat{K} . For a p -cell σ of \hat{K} , we let $\omega \cdot \sigma$ denote the *cone* of σ , which is a $(p + 1)$ -cell having cells $\{\sigma\} \cup \{\omega \cdot \tau \mid \tau \in \partial\sigma\}$ in its boundary. The *cone* $\omega \cdot L_i$ of a complex L_i consists of three parts: the vertex ω , L_i , and cones of all cells of L_i . The filtration $\hat{\mathcal{E}}$ is then defined as [12]:

$$\hat{\mathcal{E}} : L_0 \cup \{\omega\} \hookrightarrow \dots \hookrightarrow L_n \cup \{\omega\} = \hat{K} \cup \omega \cdot L_{2n} \hookrightarrow \hat{K} \cup \omega \cdot L_{2n-1} \hookrightarrow \dots \hookrightarrow \hat{K} \cup \omega \cdot L_n$$

We have that $\text{Pers}_*(\mathcal{E})$ equals $\text{Pers}_*(\hat{\mathcal{E}})$ discarding the only infinite interval [12]. Note that if a cell σ is added (to the second complex) from (\hat{K}, L_i) to (\hat{K}, L_{i-1}) in \mathcal{E} , then the cone $\omega \cdot \sigma$ is added from $\hat{K} \cup \omega \cdot L_i$ to $\hat{K} \cup \omega \cdot L_{i-1}$ in $\hat{\mathcal{E}}$.



■ **Figure 4** A Mayer-Vietoris pyramid relating the second half of \mathcal{E} and \mathcal{U} for $n = 4$.

3.4 Summary of filtration conversion

We summarize the filtration conversion process described in this section in Algorithm 3.1, in which we assume that each simplex in \mathcal{F} is given by its set of vertices. The converted standard filtration $\hat{\mathcal{E}}$ is represented by its boundary matrix D , whose columns (and equivalently the chains they represent) are treated as sets of identifiers of the boundary cells. Algorithm 3.1 also maintains the following data structures:

- `cid` denotes the map from a simplex σ to the identifier of the *most recent* copy of cell corresponding to σ .
- `del_list` denotes the list of cell identifiers deleted in the input filtration.
- `cone_id` denotes the map from the identifier of a cell to that of its coned cell.

Subroutine `CELLBOUNDARY` in Line 8 converts boundary simplices of σ_i to a column of cell identifiers based on the map `cid`. Subroutine `CONEDCELLBOUNDARY` in Line 16 returns boundary column for the cone of the cell identified by `del_id`.

We provide an example of the up-down cell-wise filtration \mathcal{U} built from a given simplex-wise filtration \mathcal{F} in Figure 5. In the example, edge c and triangle t are repeatedly added twice in \mathcal{F} , and therefore each correspond to two copies of cells in \mathcal{U} . We provide another example of a complete conversion from a given zigzag filtration to a non-zigzag filtration in Figure 6.

With the `CONVERTFILT` subroutine, Algorithm 3.2 provides a concise summary of `FASTZIGZAG`. Given that for a filtration \mathcal{F} of length m , `CONVERTFILT` takes $O(m)$ time and `CONVERTBARCODE` takes $O(1)$ time per bar, we now have the following conclusion:

► **Theorem 22.** *Given a simplex-wise zigzag filtration \mathcal{F} with length m , `FASTZIGZAG` computes $\text{Pers}_*(\mathcal{F})$ in time $T(m) + O(m)$, where $T(m)$ is the time used for computing the barcode of a non-zigzag cell-wise filtration with length m .*

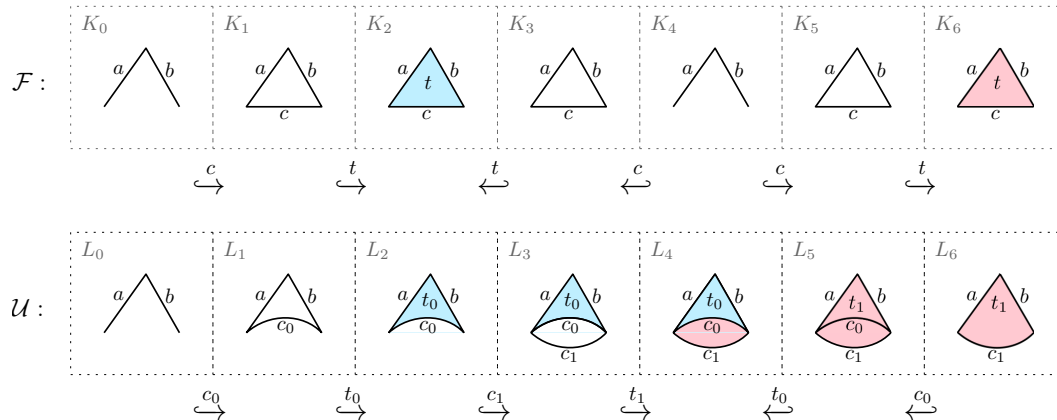
► **Remark 23.** Theoretically, $T(m) = O(m^\omega)$ [25], where $\omega < 2.37286$ is the matrix multiplication exponent [1]. So the theoretical complexity of `FASTZIGZAG` is $O(m^\omega)$.

■ **Algorithm 3.1** Pseudocode for converting input filtration.

```

1: procedure CONVERTFILT( $\mathcal{F}$ )
2:   initialize boundary matrix  $D$ , cell-id map  $\text{cid}$ , deleted cell list  $\text{del\_list}$  as empty
3:   append an empty column to  $D$  representing vertex  $\omega$  for coning
4:    $\text{id} \leftarrow 1$  ▷ variable keeping track of id for cells
5:   for each  $K_i \xleftarrow{\sigma_i} K_{i+1}$  in  $\mathcal{F}$  do
6:     if  $\sigma_i$  is being inserted then
7:        $\text{cid}[\sigma_i] = \text{id}$  ▷ get a new cell as a copy of simplex  $\sigma_i$ 
8:        $\text{col} \leftarrow \text{CELLBOUNDARY}(\sigma_i, \text{cid})$ 
9:       append  $\text{col}$  to  $D$ 
10:       $\text{id} \leftarrow \text{id} + 1$ 
11:     else
12:       append  $\text{cid}[\sigma_i]$  to  $\text{del\_list}$ 
13:   initialize map  $\text{cone\_id}$  as empty ▷  $\text{cone\_id}$  tracks id for coned cells
14:   for each  $\text{del\_id}$  in  $\text{del\_list}$  (accessed reversely) do
15:      $\text{cone\_id}[\text{del\_id}] \leftarrow \text{id}$  ▷ get a new coned cell
16:      $\text{col} \leftarrow \text{CONEDCELLBOUNDARY}(\text{del\_id}, D, \text{cone\_id})$ 
17:     append  $\text{col}$  to  $D$ 
18:      $\text{id} \leftarrow \text{id} + 1$ 
19:   return  $D$ 

```



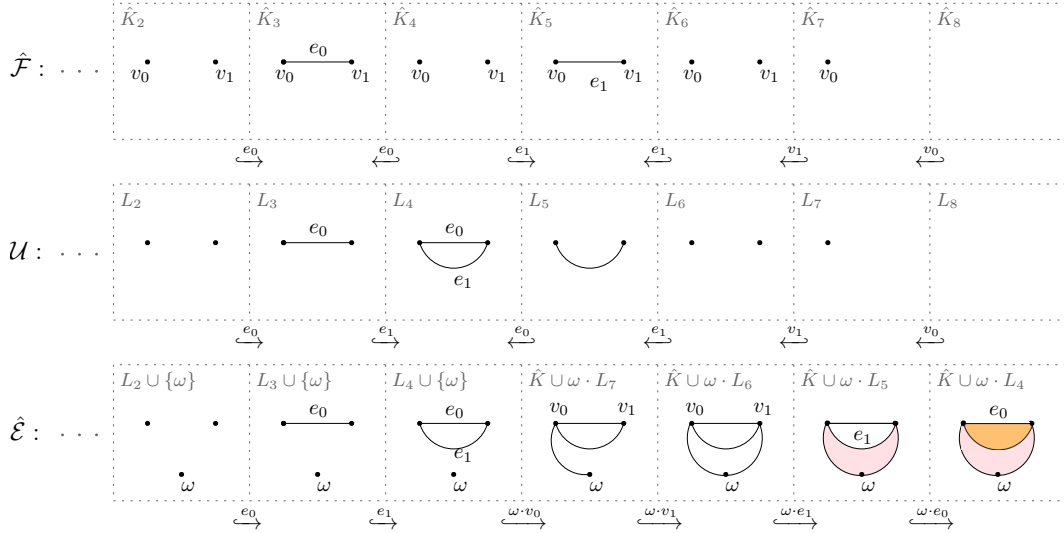
■ **Figure 5** An example of an up-down cell-wise filtration \mathcal{U} built from a given simplex-wise filtration \mathcal{F} . For brevity, \mathcal{F} does not start and end with empty complexes. The final conversion to $\hat{\mathcal{E}}$ is not shown for this example due to page-width constraint. A complete conversion for a smaller example is shown in Figure 6.

■ **Algorithm 3.2** Pseudocode for FASTZIGZAG.

```

1: procedure FASTZIGZAG( $\mathcal{F}$ )
2:    $D \leftarrow \text{CONVERTFILT}(\mathcal{F})$ 
3:    $B \leftarrow \text{COMPUTEBARCODE}(D)$ 
4:    $B' \leftarrow \text{CONVERTBARCODE}(B)$ 
5:   return  $B'$ 

```



■ **Figure 6** An example of converting a zigzag filtration $\hat{\mathcal{F}}$ to a non-zigzag filtration.

3.5 Experiments

We implement the FASTZIGZAG algorithm described in this section and compare the performance with DIONYSUS2 [26] (implementing the algorithm in [8]) and GUDHI¹ [28] (implementing the algorithm in [22, 24]). When implementing FASTZIGZAG, we utilize the PHAT [4] software for computing non-zigzag persistence. Our implementation is publicly available through: <https://github.com/taohou01/fzz>.

To test the performance, we generate eleven simplex-wise filtrations of similar lengths (5~6 millions; see Table 1). The reason for using filtrations of similar lengths is to test the impact of *repetitiveness* on the performance for different algorithms, where repetitiveness is the average times a simplex is repeatedly added in a filtration (e.g., repetitiveness being 1 means that the filtration is non-repetitive). We utilize three different approaches for generating the filtrations:

- The two non-repetitive filtrations (No. 1 and 2) are generated by first taking a simplicial complex with vertices in \mathbb{R}^3 , and then taking the height function h along a certain axis. After this, we build an up-down filtration for the complex where the first half is the lower-star filtration of h and the second half is the (reversed) upper-star filtration of h . We then randomly perform outward switches on the up-down filtration to derive a non-repetitive filtration. Note that the simplicial complex is derived from a triangular mesh supplemented by a Vietoris-Rips complex on the vertices; one triangular mesh (Dragon) is downloaded from the Stanford Computer Graphics Laboratory.
- Filtration No. 3 – 8 are generated from a sequence of edge additions and deletions, for which we then take the *clique complex* (up to a certain dimension) for each edge set in the sequence. The edge sequence is derived by randomly adding and deleting edges for a set of points.
- The remaining filtrations (No. 9 – 11) are the *oscillating Rips zigzag* [27] generated from point clouds of size 2000 – 4000 sampled from some triangular meshes (Space Shuttle from an online repository²; Bunny and Dragon from the Stanford Computer Graphics Laboratory).

¹ The code is shared by personal communication.

² Ryan Holmes: <http://www.holmes3d.net/graphics/offfiles/>

Table 1 lists running time of the three algorithms on all filtrations, where the length, maximum dimension (D), repetitiveness (Rep), and maximum complex size (MaxK) are also provided for each filtration. From Table 1, we observe that FASTZIGZAG (T_{FZZ}) consistently achieves the best running time across all inputs, with significant speedups (see column “SU” in Table 1). The speedup is calculated as the min-time of DIONYSUS2 and GUDHI divided by the time of FASTZIGZAG. Notice that since GUDHI only takes a sequence of edge additions and deletions as input (and builds clique complexes on-the-fly), we do not run GUDHI on the first two inputs in Table 1, which are only given as simplex-wise filtrations. We also observe that the speedup of FASTZIGZAG tends to be less prominent as the repetitiveness increases. This is because higher repetitiveness leads to smaller max/average complex size in the input zigzag filtration, so that algorithms directly working on the input filtration could have less processing time [8, 22, 24]. On the other hand, the complex size in the converted non-zigzag filtration that FASTZIGZAG works on is always increasing.

■ **Table 1** Running time of DIONYSUS2, GUDHI, and FASTZIGZAG on different filtrations of similar lengths with various repetitiveness. All tests were run on a desktop with Intel(R) Core(TM) i5-9500 CPU @ 3.00GHz, 16GB memory, and Linux OS.

No.	Length	D	Rep	MaxK	T_{DIO2}	T_{GUDHI}	T_{FZZ}	SU
1	5,260,700	5	1.0	883,350	2h02m46.0s	—	8.9s	873
2	5,254,620	4	1.0	1,570,326	19m36.6s	—	11.0s	107
3	5,539,494	5	1.3	1,671,047	3h05m00.0s	45m47.0s	3m20.8s	13.7
4	5,660,248	4	2.0	1,385,979	2h59m57.0s	29m46.7s	4m59.5s	6.0
5	5,327,422	4	3.5	760,098	43m54.8s	10m35.2s	3m32.1s	3.0
6	5,309,918	3	5.1	523,685	5h46m03.0s	1h32m37.0s	19m30.2s	4.7
7	5,357,346	3	7.3	368,830	3h37m54.0s	57m28.4s	30m25.2s	1.9
8	6,058,860	4	9.1	331,211	53m21.2s	7m19.0s	3m44.4s	2.0
9	5,135,720	3	21.9	11,859	23.8s	15.6s	8.6s	1.9
10	5,110,976	3	27.7	11,435	36.2s	39.9s	8.5s	4.3
11	5,811,310	4	44.2	7,782	38.5s	36.9s	23.9s	1.5

Table 2 lists the memory consumption of the three algorithms. We observe that FASTZIGZAG tends to consume more memory than the other two on the non-repetitive filtrations (No. 1 and 2) and the random clique filtrations (No. 3 – 8). However, FASTZIGZAG is consistently achieving the best memory footprint on the oscillating Rips filtrations (No. 9 – 11) despite the high repetitiveness.

4 Conclusions

In this paper, we propose a zigzag persistence algorithm called FASTZIGZAG by first treating repeatedly added simplices in an input zigzag filtration as distinct copies and then converting the input filtration to a non-zigzag filtration. The barcode of the converted non-zigzag filtration can then be easily mapped back to barcode of the input zigzag filtration. The efficiency of our algorithm is confirmed by experiments. This research also brings forth the following open questions:

■ **Table 2** Memory consumption (in gigabytes) of the three algorithms on all filtrations.

No.	Length	Rep	MaxK	M_{DIO2}	M_{GUDHI}	M_{FZZ}
1	5,260,700	1.0	883,350	3.23	–	0.59
2	5,254,620	1.0	1,570,326	3.93	–	0.61
3	5,539,494	1.3	1,671,047	15.52	13.49	9.76
4	5,660,248	2.0	1,385,979	7.64	8.43	11.04
5	5,327,422	3.5	760,098	3.27	3.40	6.22
6	5,309,918	5.1	523,685	4.94	5.27	10.23
7	5,357,346	7.3	368,830	4.03	3.91	8.19
8	6,058,860	9.1	331,211	2.12	1.48	3.68
9	5,135,720	21.9	11,859	0.92	0.47	0.50
10	5,110,976	27.7	11,435	0.88	0.48	0.47
11	5,811,310	44.2	7,782	0.95	0.60	0.51

- Parallel versions [9, 29] of the algorithms for computing standard and zigzag exist. While the computation of standard persistence in our FASTZIGZAG algorithm can directly utilize the existing parallelization techniques, we ask if the conversions done in FASTZIGZAG can be efficiently parallelized. Such an extension can provide further speedups by harnessing multi-cores.
- While persistence intervals are important topological descriptors, their *representatives* also reveal critical information (e.g., a recently proposed algorithm [16] for updating zigzag barcodes over local changes uses representatives explicitly). Can the FASTZIGZAG algorithm be adapted so that representatives for the input zigzag filtration are recovered from representatives for the converted non-zigzag filtration?

References

- 1 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.
- 2 Ulrich Bauer. Ripser: Efficient computation of vietoris–rips persistence barcodes. *Journal of Applied and Computational Topology*, 5(3):391–423, 2021.
- 3 Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Clear and compress: Computing persistent homology in chunks. In *Topological methods in data analysis and visualization III*, pages 103–117. Springer, 2014.
- 4 Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner. Phat – persistent homology algorithms toolbox. *Journal of Symbolic Computation*, 78:76–90, 2017.
- 5 Jean-Daniel Boissonnat, Tamal K. Dey, and Clément Maria. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. *Algorithmica*, 73(3):607–619, 2015. doi:10.1007/s00453-015-9999-4.
- 6 Gunnar Carlsson and Vin de Silva. Zigzag persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010.
- 7 Gunnar Carlsson, Vin de Silva, Sara Kališnik, and Dmitriy Morozov. Parametrized homology via zigzag persistence. *Algebraic & Geometric Topology*, 19(2):657–700, 2019.

- 8 Gunnar Carlsson, Vin de Silva, and Dmitriy Morozov. Zigzag persistent homology and real-valued functions. In *Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry*, pages 247–256, 2009.
- 9 Gunnar Carlsson, Anjan Dwaraknath, and Bradley J. Nelson. Persistent and zigzag homology: A matrix factorization viewpoint. *arXiv preprint*, 2019. [arXiv:1911.10693](https://arxiv.org/abs/1911.10693).
- 10 Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *Proceedings 27th European Workshop on Computational Geometry*, volume 11, pages 197–200, 2011.
- 11 Chao Chen and Michael Kerber. An output-sensitive algorithm for persistent homology. *Comput. Geom.: Theory and Applications*, 46(4):435–447, 2013. doi:10.1016/j.comgeo.2012.02.010.
- 12 David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Extending persistence using Poincaré and Lefschetz duality. *Foundations of Computational Mathematics*, 9(1):79–103, 2009.
- 13 Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Dualities in persistent (co)homology. *Inverse Problems*, 27(12):124003, 2011.
- 14 Harm Derksen and Jerzy Weyman. Quiver representations. *Notices of the AMS*, 52(2):200–206, 2005.
- 15 Tamal K. Dey and Tao Hou. Computing zigzag persistence on graphs in near-linear time. In *37th International Symposium on Computational Geometry, SoCG 2021*, volume 189 of *LIPICs*, pages 30:1–30:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 16 Tamal K. Dey and Tao Hou. Updating zigzag persistence and maintaining representatives over changing filtrations. *arXiv preprint*, 2021. [arXiv:2112.02352](https://arxiv.org/abs/2112.02352).
- 17 Tamal K. Dey, Woojin Kim, and Facundo Mémoli. Computing generalized rank invariant for 2-parameter persistence modules via zigzag persistence and its applications. In *38th International Symposium on Computational Geometry, SoCG 2022*, volume 224 of *LIPICs*, pages 34:1–34:17, 2022.
- 18 Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 454–463. IEEE, 2000.
- 19 Peter Gabriel. Unzerlegbare Darstellungen I. *Manuscripta Mathematica*, 6(1):71–103, 1972.
- 20 Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.
- 21 Petter Holme and Jari Saramäki. Temporal networks. *Physics Reports*, 519(3):97–125, 2012.
- 22 Clément Maria and Steve Y. Oudot. Zigzag persistence via reflections and transpositions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 181–199. SIAM, 2014.
- 23 Clément Maria and Steve Y. Oudot. Computing zigzag persistent cohomology. *arXiv preprint*, 2016. [arXiv:1608.06039](https://arxiv.org/abs/1608.06039).
- 24 Clément Maria and Hannah Schreiber. Discrete morse theory for computing zigzag persistence. In *Workshop on Algorithms and Data Structures*, pages 538–552. Springer, 2019.
- 25 Nikola Milosavljević, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the Twenty-Seventh Annual Symposium on Computational Geometry*, pages 216–225, 2011.
- 26 Dmitriy Morozov. *Dionysus2*. URL: <https://www.mrzv.org/software/dionysus2/>.
- 27 Steve Y. Oudot and Donald R. Sheehy. Zigzag zoology: Rips zigzags for homology inference. *Foundations of Computational Mathematics*, 15(5):1151–1186, 2015.
- 28 The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015. URL: <http://gudhi.gforge.inria.fr/doc/latest/>.
- 29 Simon Zhang, Mengbai Xiao, and Hao Wang. GPU-accelerated computation of Vietoris-Rips persistence barcodes. In *36th International Symposium on Computational Geometry (SoCG 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 30 Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

Turbocharging Heuristics for Weak Coloring Numbers

Alexander Dobler ✉

TU Wien, Austria

Manuel Sorge ✉

TU Wien, Austria

Anaïs Villedieu ✉

TU Wien, Austria

Abstract

Bounded expansion and nowhere-dense classes of graphs capture the theoretical tractability for several important algorithmic problems. These classes of graphs can be characterized by the so-called weak coloring numbers of graphs, which generalize the well-known graph invariant degeneracy (also called k -core number). Being NP-hard, weak-coloring numbers were previously computed on real-world graphs mainly via incremental heuristics. We study whether it is feasible to augment such heuristics with exponential-time subprocedures that kick in when a desired upper bound on the weak coloring number is breached. We provide hardness and tractability results on the corresponding computational subproblems. We implemented several of the resulting algorithms and show them to be competitive with previous approaches on a previously studied set of benchmark instances containing 86 graphs with up to 183831 edges. We obtain improved weak coloring numbers for over half of the instances.

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

Keywords and phrases Structural sparsity, parameterized algorithms, parameterized complexity, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.44

Related Version Based on Alexander Dobler's master thesis: <https://doi.org/10.34726/hss.2021.91580> [11]

Full Version: <https://arxiv.org/abs/2203.03358> [13]

Supplementary Material *Software:* <https://doi.org/10.5281/zenodo.5732923> [12]

Funding *Alexander Dobler:* Supported by the Vienna Science and Technology Fund (WWTF) under grant ICT19-035.

Manuel Sorge: Supported by the Alexander von Humboldt Foundation.

Anaïs Villedieu: Supported by the Austrian Science Fund (FWF) under grant P31119.

Acknowledgements We thank Nadara et al. [32] for publishing their code, some small parts of which we reused in our implementations.

1 Introduction

A *degeneracy ordering* of a graph G can be obtained by iteratively removing an arbitrary vertex of minimum degree from G and putting it in front of the current ordering [30]. The *degeneracy* of a graph is the largest degree of a vertex encountered at removal. Degeneracy orderings are immensely useful when solving various tasks on graphs both in theory and practice [8, 18, 27, 37, 22, 6]. A key observation is that many graphs in practice have small degeneracy (e.g., in the order of hundreds for millions of edges) [18]. Thus, when looking for, e.g., maximum-size cliques, it is sufficient to look within the small number of neighbors in front of each vertex in the degeneracy ordering.



© Alexander Dobler, Manuel Sorge, and Anaïs Villedieu;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 44; pp. 44:1–44:18

Leibniz International Proceedings in Informatics



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

However, degeneracy is not robust under local changes: e.g., contracting edges in a graph may arbitrarily increase its degeneracy. This property makes problems intractable on graphs that have small degeneracy if these problems are less local than finding maximum-size cliques. For example, detecting mild clique relaxations is hard on graphs of bounded degeneracy [27, 23]. Hence, we are searching for robust sparsity measures within the framework of structural sparsity [33].

We can obtain a robust variant of the class of graphs with bounded degeneracy by using the family of measures called weak r -coloring numbers [26]. For an integer r , the weak r -coloring number (wcol_r) of a graph G is the least integer k such that there is a vertex ordering with the property that, for each vertex v , there are at most k vertices u that are reachable from v by a path P of length at most r such that P does not use vertices that come before u in the ordering. The integer r , also called radius, interpolates between the degeneracy plus one ($r = 1$) and the so-called treedepth of G ($r = |V(G)|$ [20]), a measure of tree-likeness [3, 4, 25] and target of a recent implementation challenge [28].

It is important to compute the wcol_r of real-world graphs for two reasons. First, the wcol_r plays an important role in algorithmic and combinatorial techniques in structural sparsity (e.g., [2, 7, 15, 16, 17, 24, 29, 34, 36, 40]). For instance, a central concept therein is nowhere denseness and a subgraph-closed class of graphs is nowhere dense if and only if for each fixed integer r and $\epsilon > 0$ each graph G in the class has wcol_r at most $O(|V(G)|^\epsilon)$ [42]. Thus, obtaining the wcol_r s of real-world graphs will help us gauge how well this theory fits practice. Second, there is a prospect that wcol_r s will help us solve other computational problems on real-world graphs more efficiently: On nowhere-dense graph classes each problem expressible in first-order logic can be solved in near-linear time [21]. There is indeed indication that relevant classes of real-world networks, including certain scale-free networks, are nowhere dense [10]. Thus, wcol_r s may help us transfer the above theoretical near-linear-time algorithms into something practically useful. For example, this work is underway for counting subgraphs [35, 39], which is the underlying computational problem of computing graph motifs or graphlets in biological and social networks [31, 38].

Computing the wcol_r of a graph is an algorithmically challenging task: for each $r \geq 2$ it is NP-complete [20, 5]. So far, there is but one work that studies computing upper bounds for wcol_r s in real-world graphs: Nadara et al. [32] used greedy heuristics that build the associated vertex ordering by iteratively choosing a yet unordered vertex that seems favorable and putting it at the front (or the back) of the current subordering (an ordering of a subset of all vertices). Afterwards, they apply local-search techniques that make local shifts in the ordering that decrease the associated weak r -coloring number. This yields upper bounds; to date the true weak r -coloring numbers of the studied graphs are unknown, even for the smallest part of Nadara et al.'s dataset that contains graphs with 62 to 930 edges.

In this work, we study a paradigm that has previously been successfully applied to improve the quality of the results computed by greedy heuristics: turbocharging [41, 14, 1, 19]. The basic idea is that we pre-specify an upper bound $k \in \mathbb{N}$ on the wcol_r of the ordering that we want to compute. We carry out the greedy heuristic that computes the ordering iteratively. If at some point it can be detected that the ordering will yield wcol_r larger than k – the *point of regret* – we start a turbocharging algorithm. This algorithm tries to modify the current ordering, by reordering or replacing certain vertices, so as to make it possible to achieve wcol_r at most k again. Then we continue with the greedy heuristic.

Our contribution is to develop the turbocharging algorithms applied at the point of regret, obtaining running-time guarantees and lower bounds, and to implement, engineer, and evaluate these algorithms on Nadara et al.'s dataset. The two main approaches that we

study are as follows. We fix a *reconstruction parameter* $c \in \mathbb{N}$. At the point of regret, in order to obtain a subordering of wcol_r at most k we either (a) replace the last c vertices of the current subordering with new, yet unordered vertices or (b) take c vertices out of the ordering and merge them into the subordering at possibly different positions. The formal definitions are given in Section 2. We show that both approaches are NP-hard in general (see Section 3) and hence we also consider the influence of small parameters on the achievable running-time guarantees. That is, we aim to show *fixed-parameter tractability* by giving algorithms with $f(p) \cdot n^{O(1)}$ running time for a small parameter p and input size n . On the negative side, approach (a) is W[1]-hard with respect to even both c and k (Theorem 2). That is, an algorithm with running time $f(c, k) \cdot \text{poly}(n)$ is unlikely, where n is the number of vertices. This stands in contrast to Gaspers et al. [19] who obtained such an algorithm when the goal is to compute the treewidth of the input graph instead of its wcol_r . On the positive side, approach (a) is trivially tractable in polynomial time for constant c . For approach (b), we obtain a fixed-parameter algorithm with respect to $c + k$ (Theorem 4). We implemented a set of algorithms including the two previously mentioned ones and report on implementation considerations and results in Section 4. The results indicate that on average the weak r -coloring numbers achieved by Nadara et al. [32] can be improved by 5% by using our turbocharging algorithms. Using turbocharging we obtain smaller weak coloring numbers than all previous approaches on 181 of the in total 334 instances used by Nadara et al. [32].

Due to space constraints, we defer some details of proofs, implementations, and evaluations to a full version [13].

2 Preliminaries and turbocharging problems

General preliminaries. We only consider undirected, unweighted graphs G without loops. By $V(G)$ and $E(G)$ we denote the vertex set and edge set of G , respectively. For $S \subseteq V(G)$, $G[S]$ is the *induced subgraph* on vertices in S . A *path* $P = (v_1, \dots, v_n)$ in G is a non-empty sequence of vertices, such that consecutive vertices are connected by an edge. The *length* of a path is $|V(P)| - 1$. In particular, a path of length 0 consists of a single vertex. We use $\text{dist}_G(u, v)$ to denote the length of the shortest path between vertices u and v in G .

A *vertex ordering* L of a graph G is a linear ordering of $V(G)$. We write $u \prec_L v$ if vertex u precedes vertex v in L . In this case we say that u is *left of* v w.r.t. L . Equivalently, we write $u \preceq v$ if $u \prec v$ or $u = v$. We also denote vertex orderings L as sequences of its elements, that means $L = (v_1, \dots, v_n)$ represents the vertex ordering where $v_i \prec_L v_j$ iff $i < j$. We denote by $\Pi(G)$ the set of all vertex orderings of G . A *subordering* is a linear ordering of a subset $S \subseteq V(G)$. The notation L_S shall always denote a subordering where S is the set of vertices ordered in the subordering. We call the vertices in $V(G) \setminus S$ *free* with respect to L_S . Usually we will denote the set of free vertices with respect to a subordering by T . For a subordering L_S and $S' \subseteq S$ we denote by $L_S[S']$ the subordering that *agrees with* L_S on S' , that is, for all $u, v \in S'$ we have that $u \prec_{L_S[S']} v$ iff $u \prec_{L_S} v$, and all vertices in $V(G) \setminus S'$ are free w.r.t. $L_S[S']$. For a subordering L_S and $S' \supseteq S$, a subordering $L_{S'}$ is a *right extension* of L_S if $L_{S'}[S] = L_S$ and $u \prec_{L_{S'}} v$ for all $u \in S$ and $v \in S' \setminus S$. If $S' = V$, then $L_{S'}$ is called *full right extension*.

Weak coloring numbers. For a vertex ordering L of G and $r \in \mathbb{N}$, a vertex u is *weakly r -reachable* from a vertex v w.r.t. L if there exists a path P of length ℓ with $0 \leq \ell \leq r$ between u and v such that $u \preceq_L w$ for all $w \in V(P)$. Let $\text{Wreach}_r(G, L, v)$ be the set of vertices that are weakly r -reachable from v in G w.r.t. L . The *weak r -coloring number* $\text{wcol}_r(G, L)$ of a vertex ordering L is $\text{wcol}_r(G, L) = \max_{v \in V(G)} |\text{Wreach}_r(G, L, v)|$, and the weak r -coloring number $\text{wcol}_r(G)$ of G is $\text{wcol}_r(G) = \min_{L \in \Pi} \text{wcol}_r(G, L)$.

Turbocharging. Our goal is to find a vertex ordering L of a given graph G with small $\text{wcol}_r(G, L)$ by applying turbocharging. We start with two well-known iterative greedy heuristics (descriptions will follow) of Nadara et al. [32] that build vertex orderings with small weak r -coloring number from left to right. That is, these heuristics start with the empty subordering $L_S = \emptyset$ and in each step compute a right extension of L_S that contains one more vertex. This process is continued until the constructed subordering contains all vertices. A key observation about this process that we can use for turbocharging is that the size of the weakly reachable set of each vertex cannot decrease:

► **Observation 1.** *Let L_S be a subordering, $u, v \in V(G)$, and L be a full right extension of L_S . If $u = v$, or $u \in S$ and there exists a path P of length ℓ with $0 \leq \ell \leq r$ between u and v such that $u \preceq_{L_S} w$ for all $w \in V(P) \cap S$, then $u \in \text{Wreach}_r(G, L, v)$.*

For u and v as in Observation 1 we extend the definition of weak r -reachability to suborderings L_S by defining $u \in \text{Wreach}_r(G, L_S, v)$ and $\text{wcol}_r(G, L_S) = \max_{v \in V(G)} |\text{Wreach}_r(G, L_S, v)|$. We immediately obtain that $\text{wcol}_r(G, L_S)$ is a lower bound; that is, if L is a full right extension of L_S (such as obtained by one of the heuristics), then $\text{wcol}_r(G, L) \geq \text{wcol}_r(G, L_S)$.

The two heuristics of Nadara et al. that we apply are:

- The Degree-Heuristic: This heuristic orders vertices by descending degree, ties are broken arbitrarily.
- The Wreach-Heuristic: For a subordering L_S , this heuristic picks the free vertex v with the largest $\text{Wreach}_r(G, L_S, v)$. Ties are broken by descending degree.

Nadara et al. proposed several other heuristics, but those heuristics do not build vertex orderings from left to right, but in different orders. Additionally, the above heuristics are among the best-performing ones with regard to computed weak coloring numbers and runtime.

In what follows, let us assume that we want to compute a vertex ordering L of a graph G with $\text{wcol}_r(G, L) \leq k$ where $k \in \mathbb{N}$. We might apply one of the heuristics until we obtain a subordering L_S such that $\text{wcol}_r(G, L_S) > k$. We call such a subordering *non-extendable* (and otherwise the subordering is *extendable*); we also say that this point in the execution of the heuristic is the *point of regret*. We then consider two exact *turbocharging problems* that try to locally augment L_S , such that it is extendable again. If the *turbocharging algorithms* for these problems that we later propose are successful in making L_S extendable, then we continue applying the heuristic until we have to repeat this process (trying to find a vertex order L with $\text{wcol}_r(G, L) \leq k$).

Motivated by a turbocharging algorithm for computing tree-decompositions by Gaspers et al. [19] we consider replacing a bounded-length suffix of the current subordering. That is, we specify a *reconstruction parameter* $c \in \mathbb{N}$ in advance and, at the point of regret, we remove the last c vertices from L_S and then try to add c (possibly) different vertices. This leads to the following turbocharging problem.

INCREMENTAL CONSERVATIVE WEAK r -COLORING (IC-WCOL(r))

Instance: A graph G , a subordering L_S , and positive integers k and c .

Question: Is there an extendable right extension $L_{S'}$ of L_S such that $|S' \setminus S| = c$?

Our second turbocharging approach is based on a vertex v with $|\text{Wreach}_r(G, L_S, v)| > k$. Therein, instead of the suffix of the current order, we choose a set S_2 of vertices related to the weakly r -reachable set of v (details follow in Section 4). We remove the vertices in S_2 from L_S , leaving us with the subordering L_{S_1} , and then we try to reinsert the vertices in S_2 while decreasing the weak coloring number.

WCOL-MERGE(r)

Instance: A graph G , an integer k , two disjoint sets S_1 and S_2 such that $S_1, S_2 \subseteq V(G)$, and a subordering L_{S_1} .

Question: Is there an extendable subordering $L_{S_1 \cup S_2}$ such that $L_{S_1 \cup S_2}[S_1] = L_{S_1}$?

Herein, we put the *reconstruction parameter* c equal to $|S_2|$ and denote by T the set of free vertices $V(G) \setminus (S_1 \cup S_2)$.

3 Algorithms and running-time bounds

We continue by providing algorithmic upper and lower bounds for INCREMENTAL CONSERVATIVE WEAK r -COLORING (IC-WCOL(r)) and WCOL-MERGE(r), starting with IC-WCOL(r).

3.1 IC-WCOL(r)

The first theoretical result that we want to present is the NP-hardness of INCREMENTAL CONSERVATIVE WEAK r -COLORING for each $r \geq 1$ by giving a reduction from INDEPENDENT SET. INDEPENDENT SET takes as input a graph G and a positive integer p and asks if there is a set of vertices I of size at least p such that $\{u, v\} \not\subseteq I$ for all $\{u, v\} \in E(G)$.

► **Theorem 2.** *For each fixed $r \geq 1$, INCREMENTAL CONSERVATIVE WEAK r -COLORING is NP-hard and $W[1]$ -hard when parameterized by $k + c$.*

The proof can be found in the full version of this paper [13] and gives a polynomial reduction from of INDEPENDENT SET which is NP-complete to an instance of IC-WCOL(r). The idea is that the parameter p – the desired independent set size of a given INDEPENDENT SET instance – is transformed to the parameters $c = p$ and $k = 2$ of IC-WCOL(r), giving us a parameterized reduction from INDEPENDENT SET to IC-WCOL(r). INDEPENDENT SET is $W[1]$ -hard when parameterized by p , and thus we obtain the stated $W[1]$ -hardness.

On the other hand, it is not hard to see that INCREMENTAL CONSERVATIVE WEAK r -COLORING is in XP: We can simply try placing any of the vertices from $V(G) \setminus S$ into the next free position right of L_S . As there are c free positions the overall algorithm runs in $\mathcal{O}(|V(G) \setminus S|^c \cdot |V(G)|^{\mathcal{O}(1)}) \subseteq \mathcal{O}(|V(G)|^c \cdot |V(G)|^{\mathcal{O}(1)})$ time.

► **Proposition 3.** *INCREMENTAL CONSERVATIVE WEAK r -COLORING parameterized by the reconstruction parameter c is in XP.*

Our algorithm for INCREMENTAL CONSERVATIVE WEAK r -COLORING is based on Proposition 3, we will go into more detail in Section 4.

3.2 WCOL-Merge(r)

It is easy to see that WCOL-MERGE(r) is NP-hard for $r \geq 2$: Given a graph G , we can decide $\text{wcol}_r(G) \leq k$ by creating an instance (H, S_1, S_2, L_{S_1}) of WCOL-MERGE(r), setting $S_1 = L_{S_1} = \emptyset$, $H = G$, and $S_2 = V(G)$. As deciding $\text{wcol}_r(G) \leq k$ is NP-hard for $r \geq 2$ [20, 5], so is WCOL-MERGE(r). On the positive side, we can show fixed-parameter tractability of WCOL-MERGE(r) parameterized by k and $|S_2|$.

► **Theorem 4.** *WCOL-MERGE(r) is solvable in time $\mathcal{O}(|S_2|! \cdot k^{|S_2|} \cdot |V(G)|^{\mathcal{O}(1)})$. In particular, WCOL-MERGE(r) is fixed-parameter tractable when parameterized by $k + |S_2|$.*

Intuitively, the algorithm behind Theorem 4 tries to place each vertex v in S_2 one by one by trying all relevant positions. The key insight is that only few positions are relevant (called breakpoints below). Namely those positions that correspond to vertices that are weakly r -reachable from v when placed at the end of the ordering. As only few vertices can be reachable from v when placed in the correct position, we only need to try the first k corresponding positions.

To describe the algorithm we need definitions for two operations that we use throughout.

► **Definition 5.** Let G be a graph, $L_S = (s_1, \dots, s_n)$ a subordering, and $v \in V(G) \setminus S$. We denote by $\text{placeafter}(L_S, s_i, v)$ the subordering of vertices $S \cup \{v\}$ that is obtained by placing v directly after s_i . To be precise, $\text{placeafter}(L_S, s_i, v) := (s_1, \dots, s_i, v, s_{i+1}, \dots, s_n)$. Equivalently, $\text{placebefore}(L_S, s_i, v) := (s_1, \dots, s_{i-1}, v, s_i, \dots, s_n)$.

This leads to the definitions of breakpoints, which are crucial for the proof of Theorem 4.

► **Definition 6.** Let G be a graph, $L_S = (s_1, \dots, s_n)$ a subordering, and $v \in V(G) \setminus S$. A vertex $s \in S$ is called *breakpoint* of v if $\text{Wreach}_r(G, \text{placebefore}(L_S, s, v), v) \neq \text{Wreach}_r(G, \text{placeafter}(L_S, s, v), v)$. Let $\text{bp}(G, L_S, v) \subseteq S$ be the set of breakpoints of v .

We also notice another useful property of breakpoints, which is proved in the full version of this paper [13].

► **Lemma 7.** Let $v \in V(G) \setminus S$. We have $s \notin \text{bp}(G, L_S, v)$ if and only if for all $u \in V(G)$

$$\text{Wreach}_r(G, \text{placebefore}(L_S, s, v), u) = \text{Wreach}_r(G, \text{placeafter}(L_S, s, v), u).$$

If we add a vertex v to a subordering L_S to obtain a new subordering $L_{S \cup \{v\}}$, then the weakly reachable vertices $\text{Wreach}_r(G, L_{S \cup \{v\}}, v)$ consist of v and a subset of $\text{bp}(G, L_S, v)$. We formalize this as follows.

► **Lemma 8.** Let G be a graph, $L_S = (s_1, \dots, s_n)$ be a subordering, and $v \in V(G) \setminus S$. Furthermore, let $L_{S \cup \{v\}}$ be a subordering such that $L_{S \cup \{v\}}[S] = L$. Then

$$\text{Wreach}_r(G, L_{S \cup \{v\}}, v) \setminus \{v\} = \{s \in \text{bp}(G, L_S, v) \mid s \preceq_{L_{S \cup \{v\}}} v\}.$$

Proof. Let X be the set $\{s \in \text{bp}(G, L_S, v) \mid s \preceq_{L_{S \cup \{v\}}} v\}$, we prove both inclusions of the equation $\text{Wreach}_r(G, L_{S \cup \{v\}}, v) \setminus \{v\} = X$.

Assume that $s \in (\text{Wreach}_r(G, L_{S \cup \{v\}}, v) \setminus \{v\})$. As s is weakly r -reachable from v , there is a path $P = (v, u_1, \dots, u_\ell, s)$ of length of at most r that does not go left of s w.r.t. to $L_{S \cup \{v\}}$. Consider the same path P in $\text{placeafter}(L_S, s, v)$. Clearly, s is also weakly r -reachable in this subordering because of the same path. Contrary to that, s cannot be weakly r -reachable from v in $\text{placebefore}(L_S, s, v)$ because v is left of s in that subordering. Hence, $s \in \text{bp}(G, L_S, v)$ and $\text{Wreach}_r(G, L_{S \cup \{v\}}, v) \setminus \{v\} \subseteq X$.

Assume that $s \in X$. Then s must be weakly r -reachable from v w.r.t. $\text{placeafter}(L_S, s, v)$ through a path P of length at most r . But s is also weakly r -reachable from v w.r.t. $L_{S \cup \{v\}}$ through the same path P . Hence, $X \subseteq \text{Wreach}_r(G, L_{S \cup \{v\}}, v) \setminus \{v\}$ also holds. ◀

Using the above tools, we can now formally describe Algorithm 1, which obtains the stated runtime of Theorem 4 and is given as a recursive function `RECURSIVE-MERGE`. As alluded to before, the intuition is that for each vertex $v \in S_2$ we only have to consider placing it before its breakpoints w.r.t. L_{S_1} . As the breakpoints of a vertex will be in its weakly r -reachable set, only the leftmost k breakpoints are relevant. A detailed proof of the correctness and the runtime can be found in the full version of this paper [13].

■ **Algorithm 1** Recursive FPT-algorithm for $\text{WCOL-MERGE}(r)$.

```

1 Recursive-merge( $S_1, S_2, T, L_{S_1}$ ):
2   if  $|S_2| = 0 \wedge \forall v \in V(G) : |\text{Wreach}_r(G, L_{S_1}, v)| \leq k$  then return  $L_{S_1}$ ;
3   for  $v \in S_2$  do
4     for  $s \in \text{bp}(G[S_1 \cup T \cup \{v\}], L_{S_1}, v)$  do
5        $L_{S_1 \cup \{v\}} \leftarrow \text{placebefore}(L_{S_1}, s, v)$ ;
6       if  $|\text{Wreach}_r(G[S_1 \cup T \cup \{v\}], L_{S_1 \cup \{v\}}, v)| \leq k$  then
7         answer  $\leftarrow \text{Recursive-merge}(S_1 \cup \{v\}, S_2 \setminus \{v\}, T, L_{S_1 \cup \{v\}})$ ;
8         if answer  $\neq$  false then return answer;
9       end
10    end
11     $s \leftarrow$ rightmost vertex of  $S_1$  w.r.t.  $L_{S_1}$ ;
12     $L_{S_1 \cup \{v\}} \leftarrow \text{placeafter}(L_{S_1}, s, v)$ ;
13    if  $|\text{Wreach}_r(G[S_1 \cup T \cup \{v\}], L_{S_1 \cup \{v\}}, v)| \leq k$  then
14      answer  $\leftarrow \text{Recursive-merge}(S_1 \cup \{v\}, S_2 \setminus \{v\}, T, L_{S_1 \cup \{v\}})$ ;
15      if answer  $\neq$  false then return answer;
16    end
17  end
18  return false

```

4 Implementation and experiments

This section contains implementation details for the heuristics and both turbocharging algorithms. Furthermore, we give the experimental setup and experimental results.

4.1 Algorithm implementations and heuristic improvements

Interesting details are omitted from the algorithmic results of Section 3. We want to give some implementation details for $\text{IC-WCOL}(r)$ and $\text{WCOL-MERGE}(r)$, and how the algorithms for these problems are combined with the heuristics. Additionally, we give a third turbocharging approach called $\text{IC-WCOL-RL}(r)$ in this section.

In our implementations of heuristics and turbocharging algorithms we store and update the current subordering L_S in a simple array. We also store and update the sets $\text{Wreach}_r(G, L_S, v)$ and $\text{Wreach}_r^{-1}(G, L_S, v) = \{w \in V(G) : v \in \text{Wreach}_r(G, L_S, w)\}$ (see below for their usage). Updating weakly r -reachable sets during placements and removals of vertices v is done by computing the set $\text{Wreach}_r^{-1}(G, L_S, v)$ via a breadth-first search that respects the order L_S , and updating the corresponding weakly r -reachable sets.

Additionally, we slightly adapt the Degree-Heuristic: We aim for vertex orderings L with $\text{wcol}_r(G, L) \leq k$. Consider a subordering L_S that was created by the heuristic and needs to be extended. To obtain weak coloring number k it would intuitively make sense to place a free vertex v with $\text{wcol}_r(G, L_S) = k$ immediately to the right of that subordering s.t. its weakly r -reachable set cannot increase anymore. This is indeed always correct – if there is a full right extension L of L_S with $\text{wcol}_r(G, L) \leq k$, then there is another one that starts by placing v immediately to the right of L_S (for a formal statement and a proof we refer to the full version of this paper [13]). In our implementation of the Degree-Heuristic we apply this observation and place such a vertex v immediately. The Wreach-Heuristic does this implicitly.

We continue by explaining individual details for IC-WCOL(r) and WCOL-MERGE(r), and how they are applied to a non-extendable subordering L_S with free vertices T .

IC-WCOL(r). We have implemented the XP-algorithm for INCREMENTAL CONSERVATIVE WEAK r -COLORING as outlined in Proposition 3. Given a subordering L_S , we have to extend L_S to the right by c vertices, that means that we have c positions to fill. We implement a search tree algorithm that fills these positions from left to right recursively. That is, in a search tree node we try all possibilities of placing a free vertex into the leftmost free position i and recurse into search tree nodes that try placing the remaining free vertices into position $i + 1$, and so on, until all c positions are filled.

If after a placement of a vertex we obtain a non-extendable subordering, we can cut off this branch of the search tree, as weakly r -reachable sets of vertices can only increase in this branch. We also store edges of $G[T]$ separately as an array of hash sets. This enables, in a search tree node, to update the sets $\text{Wreach}_r^{-1}(G, L_S, v)$ on placement/removal by a simple depth- r breadth-first-search in $G[T]$. This decreases the number of enumerated edges compared to the trivial approach.

Free vertices T are placed into a position i in a specific order inside a search tree node: let L_S be the non-extendable subordering that triggered turbocharging and let v be the rightmost vertex of L_S . We try placing $u_1 \in T$ before $u_2 \in T$ into the free slot i if $\text{dist}_G(u_1, v) < \text{dist}_G(u_2, v)$. Preliminary experiments suggested that this order is preferable to a random one. We compute $\text{dist}_G(u, v)$ for all u, v with Johnson's Algorithm [9] for sparse graphs once in the beginning.

WCOL-Merge(r). We apply WCOL-MERGE(r) to a non-extendable subordering L_S in the following way. Let $U = \{v \in V(G) : |\text{Wreach}_r(G, L_S, v)| > k\}$ be the set of *overfull* vertices. Let c be a positive integer and let X be a random subset of $\bigcup_{v \in U} \text{Wreach}_r(G, L_S, v)$ of size $\min(c, |\bigcup_{v \in U} \text{Wreach}_r(G, L_S, v)|)$. If the size of X is less than c , we randomly add additional vertices from $V(G)$ to X , until the size of X is c . We try to fix L_S by defining an instance of WCOL-MERGE(r). Namely, we set $S_1 = S \setminus X$, $S_2 = X$, and $L_{S_1} = L_S[S_1]$. We then solve this instance using Algorithm 1. By Theorem 4 we obtain a turbocharging algorithm that has fixed-parameter tractable running time when parameterized by the desired coloring number k and reconstruction parameter c .

In our implementation we apply WCOL-MERGE(r) multiple times with different randomly selected sets X as defined above, until we obtain an extendable subordering. Preliminary experiments showed that choosing the whole set $\bigcup_{v \in U} \text{Wreach}_r(G, L_S, v)$ as X leads to timeouts often whereas random subsets still allowed us to fix L_S . If we do not find an extendable subordering after the 10th application of WCOL-MERGE(r), we report that turbocharging was not successful.

We now discuss the implementation of Algorithm 1. Consider the vertex v in Algorithm 1. We only have to iterate over the k leftmost breakpoints of v due to Lemma 8, which can be easily done by storing and maintaining $\text{Wreach}_r^{-1}(G[S_1 \cup T \cup \{v\}], L_{S_1 \cup T}, v)$. Let s be a breakpoint of v and let $L_{S_1 \cup \{v\}} = \text{placeafter}(L_{S_1}, s, v)$. The leftmost $s' \in \text{Wreach}_r^{-1}(G[S_1 \cup T \cup \{v\}], L_{S_1 \cup \{v\}}, v)$ that is not v is the next possible breakpoint of v . Additionally, we do not need to recurse if the size of some set $\text{Wreach}_r(G[S_1 \cup T], L_{S_1}, v)$ exceeds k for some $v \in S_1 \cup T$, as these sets can only increase in subsequent recursion calls.

We also know that subsets of some weakly r -reachable sets of vertices T are already fixed. Namely, for all vertices v in the r -neighborhood of u in $G[T \cup \{u\}]$ with $u \in S_2$, vertex u will always be in the weakly r -reachable set of v if u is placed somewhere into the subordering L_{S_1} . We take this into account when calculating lower bounds for the sizes of weakly r -reachable sets of vertices in T (and break the search if they exceed size k).

IC-WCOL-RL(r). We also implemented a turbocharging algorithm that is not discussed above. It is based on the *Sreach-Heuristic*, which builds a vertex ordering of low weak coloring number from right to left (instead of from left to right as above) [32]. It starts with an empty subordering L_S and, during each step, the heuristic adds to the left front of L_S a free vertex v that minimizes the number of so-called potentially strongly r -reachable vertices after being placed. Herein, a vertex u is *potentially strongly r -reachable* w.r.t. L_S from a vertex $v \in S$ if $u \in \text{Wreach}_r(G[S], L_S, v)$ or there is a path P of length at most r from v to u in G such that $V(P) \cap T = \{u\}$. It can be shown (refer to a formal statement and a proof in the full version of this paper [13]) that the set of potentially strongly r -reachable vertices of v only grows when extending L_S to the left and that, when $S = V(G)$, this set equals $\text{Wreach}_r(G, L_S, v)$. Thus, we define a *point of regret* for this heuristic as a point in the execution where there is a vertex v such that the size of its potentially strongly r -reachable set exceeds the desired weak coloring number k . Accordingly, we say that L_S is *non-extendable*, and otherwise it is *extendable*. We then solve the turbocharging problem in which we aim to replace the c leftmost vertices of L_S with arbitrary free vertices in order to make L_S extendable again. We do this using a search-tree algorithm analogous to IC-WCOL(r). We also use the above turbocharging approach with a heuristic that chooses the next vertex among the free vertices based on the smallest degree. We call this heuristic Degree-Heuristic¹.

We tried further heuristic optimizations, mainly for the left-to-right approaches, based on lower bounds (for early search termination), guided branching (towards faster decomposition into trivial instances), and ordered adjacency lists (to speed up computation of weakly reachable sets) but they did not improve the resulting coloring numbers.

4.2 Experiments setup

Computation environment. All experiments were performed on a cluster of 20 nodes. Each node is equipped with two Intel Xeon E5-2640 v4, 2.40GHz 10-core processors and 160 GB RAM. The optimization for each instance and an algorithm was pinned to a specific core of a cluster node (simultaneous multithreading was disabled). All implementations of heuristics and turbocharging algorithms were done in C++17, and made use of the Boost library², version 1.77.0. The code was compiled on Linux with g++ version 7.5.0 and with the flags `-std=c++17 -O2`. The optimization process (see *application of turbocharging* below) that calls the heuristics combined with the turbocharging algorithms (implemented in C++) was implemented in Python3 and executed with Python 3.7.13. A memory limit of 16 GB was set (a process only starts if the required memory is free). The source code is available online [12].

Instances. Each instance in our data set is a tuple consisting of a graph G and a radius $r \in \mathbb{N}$. The radii r are between 2 and 5, as also used by Nadara et al. [32]. The graphs G form a subset of the graphs used by Nadara et al. This enables us to use weak coloring numbers of orderings computed by Nadara et al. as a baseline. Furthermore, we can compare for a heuristic, the improvement achieved by the local search of Nadara et al. to the improvement achieved by our turbocharging algorithms.

The graphs consist of real-world data, the PACE 2016 Feedback Vertex Set problems, random planar graphs, and random graphs with bounded expansion. Nadara et al. classified the graphs into four classes based on the number of edges – small (up to 1k edges), medium (up to 10k edges), big (up to 48k edges), and huge. For a detailed explanation and references

¹ The name is the same as in the left-to-right setting; there will be no confusion between the two because the direction will be clear from the context.

² <https://www.boost.org/>

■ **Algorithm 2** Algorithm for iteratively decreasing the weak coloring number by turbocharging.

Input: A graph G , an integer r , a heuristic \mathcal{H} , and a turbocharged heuristic $TC\text{-}\mathcal{H}$
Output: An ordering L of vertices V

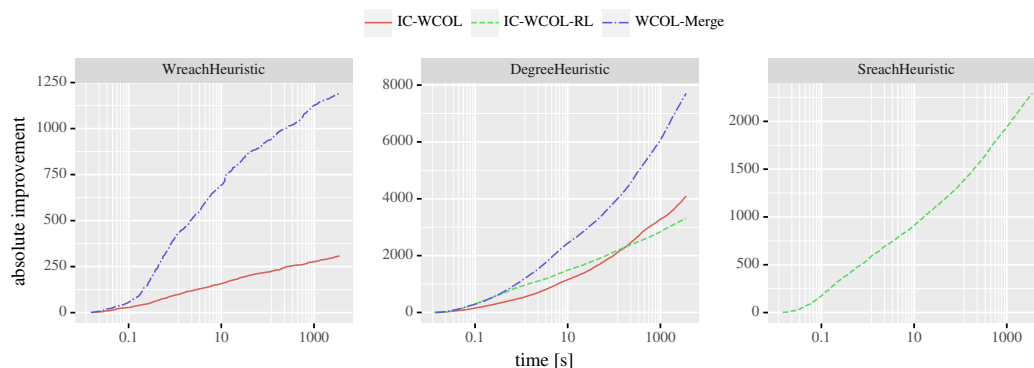
- 1 $L \leftarrow$ ordering of vertices V computed by the heuristic \mathcal{H} ;
- 2 $k \leftarrow \text{wcol}_r(G, L)$;
- 3 Start a timer; after t seconds abort the program, and **return** the current value of L
- 4 **while true do**
- 5 $c \leftarrow 1$;
- 6 **while true do**
- 7 Try to compute an ordering of vertices V with weak r -coloring number $k - 1$
 using $TC\text{-}\mathcal{H}$ with reconstruction parameter c ;
- 8 If successful, assign this ordering to L , set $k \leftarrow \text{wcol}_r(G, L)$, and **break**;
- 9 Otherwise, set $c \leftarrow c + 1$;
- 10 **end**
- 11 **end**

for all input graphs we refer to Nadara et al. [32]; the instances are available online³. We considered all instances except those where one of the three heuristics of Nadara et al. that we also consider did not yield a result (the Degree-Heuristic, Wreach-Heuristic, and Sreach-Heuristic). That is, they timed out after 300 seconds or ran into a memory limit (16 GB). In total, our dataset contains 334 instances.

Application of turbocharging. Our application of turbocharging with a heuristic \mathcal{H} works as follows. Given a graph G and a radius r , we start with a run of \mathcal{H} without turbocharging to produce a vertex ordering for G with a baseline weak r -coloring number k . We then start a timer that runs for t seconds, aborting the rest of the algorithm when it terminates. We then iteratively decrease k and apply \mathcal{H} together with the turbocharging approach to try and find a vertex ordering for G with weak r -coloring number at most k . In each such try, we start with the reconstruction parameter $c = 1$. If no ordering with the desired weak r -coloring number was produced by the turbocharged heuristic, we increase c by one and try again. If an ordering with weak r -coloring number $k' \leq k$ was produced, we set $k = k' - 1$ and repeat the process. The precise algorithm is given in Algorithm 2.

For our experiments we applied all compatible combinations of heuristics and the turbocharged versions to each instance. Furthermore, we computed orderings for radii ranging from 2 to 5, motivated by Nadara et al. who used the same values. We run all experiments twice, once with timeout $t = 300s$ and once with $t = 3600s$. Results with $t = 300s$ are directly compared with the results of Nadara et al. who used 300 seconds as timeout. Results with $t = 3600s$ give us the ability to investigate the potential of turbocharging over longer periods of time.

³ <https://kernelization-experiments.mimuw.edu.pl/>



■ **Figure 1** Line plot of the cumulative absolute improvements over time achieved by the turbocharging approaches broken down by the underlying heuristics. The x -axis (time) is scaled logarithmically.

4.3 Results

We now show to which extent our turbocharging approaches improve the results of heuristics, compare the achieved weak coloring numbers to the ones of Nadara et al., and provide observations about the performance of turbocharging.

Impact of turbocharging. Turbocharging has the advantage that investing gradually more time will yield gradually better results – setting the reconstruction parameter to $c = |V(G)|$, we (in theory) even can provably obtain the optimum. Figure 1 shows the cumulative sum of absolute improvements over time when comparing each turbocharged heuristic with the underlying plain heuristic. That is, for a specific time t , the cumulative sum of absolute improvements for a turbocharging algorithm \mathcal{A} and a heuristic \mathcal{H} is $\sum_{I \in \text{instances}} (k_{I, \mathcal{H}} - k_{I, \mathcal{A}, \mathcal{H}, t})$, where $k_{I, \mathcal{H}}$ is the coloring number achieved by the heuristic and $k_{I, \mathcal{A}, \mathcal{H}, t}$ is the coloring number achieved by the turbocharged heuristic after time t . Note that the y -axes do not have the same ranges as the underlying heuristics have different performance levels. All plots exhibit logarithmic or similar to logarithmic growth, which means that the gained absolute improvement is approximately logarithmic in the invested time in the most cases. WCOL-MERGE(r) clearly yields faster and larger improvements than IC-WCOL(r), and it also supersedes IC-WCOL-RL(r) for the Degree-Heuristic. One reason may be that WCOL-MERGE(r) is fixed-parameter tractable and the associated parameters are small.

Further evaluations of the executions of turbocharging algorithms are analysed in detail in the full version of this paper [13]. Some observations therein are that the number of applications of turbocharging per run of a heuristic ranges in the order of at most hundreds for IC-WCOL(r) and WCOL-MERGE(r) and on average in the single digits; for IC-WCOL-RL(r) these numbers are one to two orders of magnitude larger. Successful applications of a turbocharged heuristic (those where the weak coloring number could be improved) mostly only use very little time and search tree nodes, and have small reconstruction parameters – mostly $c = 1$. That is, it is mostly the case that a heuristic wants to place a vertex that is “suboptimal”, while placing nearly any other vertex will achieve lower weak coloring number. The fraction of time spent on turbocharging is also low, which means that most of the time when we can improve the weak coloring number achieved by a heuristic, this can be done easily and in little time.

■ **Table 1** IC-WCOL(r): White columns give relative improvements, light gray columns give quality ratios, dark gray columns give average/maximum absolute improvements. For improvements, we compare to the underlying heuristic without turbocharging and without local search. Time limit: 300s.

tests	r	Wreach IC-WCOL(r)		Degree IC-WCOL(r)			
small	2	-6.2%	0.8/5	7.2%	-7.3%	3.2/14	18.4%
	3	-7.3%	1.0/6		-9.3%	4.3/20	
	4	-11.3%	1.7/7		-11.2%	4.2/15	
	5	-15.8%	1.9/8		-16.8%	3.9/12	
medium	2	-5.6%	0.6/2	3.7%	-7.1%	5.0/14	17.4%
	3	-9.2%	1.0/11		-9.7%	9.0/35	
	4	-11.6%	0.3/2		-15.8%	9.9/31	
	5	-16.8%	1.3/15		-20.6%	9.9/41	
big	2	-9.6%	0.1/1	0.7%	-21.6%	7.2/31	12.0%
	3	-9.5%	0.4/3		-20.8%	15.3/47	
	4	-12.7%	0.3/2		-32.5%	14.5/42	
	5	-38.3%	0.2/1		-30.4%	13.7/38	
huge	2	-2.0%	0.1/1	0.2%	-39.1%	2.8/16	0.9%
	3	-21.4%	0.1/1		-35.0%	1.9/6	
	4	-6.9%	0.0/0		-29.9%	1.8/5	
	5	-8.9%	0.3/1		-16.5%	1.7/4	

By dataset group and radius. Next, we fix a time threshold of 300s (same as Nadara et al.) that we might reasonably invest in practice for computing weak coloring numbers. We present the improvements in weak coloring numbers gained by turbocharging over the plain heuristics broken down according to the instance group (small, medium, big, huge) and the radius r . We again provide absolute improvements when comparing with the underlying heuristic, and we also consider the average *relative improvement* of the weak r -coloring number when comparing the turbocharged heuristic to the plain heuristic; that is, the relative improvement is $1 - k_{I,\mathcal{A},\mathcal{H},t}/k_{I,\mathcal{H}}$ for $t = 300s$. For each turbocharging algorithm we show results for both turbocharged heuristics. For IC-WCOL(r) and WCOL-MERGE(r) these are the Wreach- and Degree-Heuristic, and for IC-WCOL-RL(r) these are the Sreach- and Degree-Heuristic. The results are given in three tables corresponding to the three turbocharging approaches.

We furthermore compare the achieved coloring numbers of each approach to the best coloring numbers computed by Nadara et al.: For each instance I , let $\text{bestNadara}(I)$ be the smallest weak r -coloring number of an ordering of vertices of instance I achieved by an approach of Nadara et al. Note that they implemented seven different heuristics and for each computed ordering they applied a local search to iteratively reduce the weak r -coloring number. To evaluate one of our approaches, we take the weak r -coloring number $k_{I,\mathcal{A},\mathcal{H},t}$ for instance I obtained by our approach for $t = 300s$ and compute the average $1 - k_{I,\mathcal{A},\mathcal{H},t}/\text{bestNadara}(I)$ (in percent) taken over all instances I in the corresponding data set. We call this value *quality ratio*. Note that positive values mean that the approach achieves lower weak coloring numbers on average when compared to the best weak coloring numbers achieved by Nadara et al.

■ **Table 2** WCOL-MERGE(r): White columns give relative improvements, light gray columns give quality ratios, dark gray columns give average/maximum absolute improvements. For improvements, we compare to the underlying heuristic without turbocharging and without local search. Time limit: 300 s.

tests	r	Wreach	WCOL-MERGE(r)		Degree	WCOL-MERGE(r)	
small	2	-1.0%	1.4/5	19.1%	0.3%	4.1/11	33.7%
	3	3.3%	2.8/8		2.9%	6.3/25	
	4	0.7%	4.2/11		2.3%	7.2/23	
	5	-1.2%	5.4/14		-0.2%	8.0/19	
medium	2	2.4%	1.6/7	15.0%	1.1%	6.4/16	32.7%
	3	0.4%	2.7/15		0.6%	11.6/46	
	4	-0.2%	2.9/13		-1.9%	13.3/36	
	5	-5.3%	4.2/16		-5.5%	15.5/50	
big	2	-0.3%	1.7/7	9.4%	-7.5%	11.0/32	24.2%
	3	-1.8%	2.1/9		-8.1%	23.4/65	
	4	-4.2%	2.8/11		-19.3%	26.6/63	
	5	-27.2%	4.9/26		-20.5%	26.6/67	
huge	2	-0.6%	1.7/5	1.0%	-14.9%	28.4/84	12.2%
	3	-20.5%	1.8/5		-21.9%	27.4/49	
	4	-6.7%	0.5/2		-27.6%	11.2/20	
	5	-8.7%	1.0/2		-15.5%	6.0/14	

In Table 1 we present the performance of the IC-WCOL(r) approach. It is evident that the relative and absolute improvements achieved for the Degree-Heuristic is significantly higher than for the Wreach-Heuristic, although this is partly due to the fact that the Degree-Heuristic achieves worse results than the Wreach-Heuristic before turbocharging. Relative and absolute improvements decrease for larger instances.

Table 2 contains the results for WCOL-MERGE(r). Here, turbocharging achieves positive quality ratios for some instance classes and radii. The relative and absolute improvements are much larger than for IC-WCOL(r), especially for the huge instances and the Degree-Heuristic. It is also interesting that while the Degree-Heuristic generally computes orderings of higher weak coloring number than the Wreach-Heuristic, the turbocharged version of the Degree-Heuristic computes orderings of similar or even lower weak coloring numbers than the turbocharged version of the Wreach-Heuristic for the small and medium instances. We do not see an obvious reason for that, but it could again indicate the power of the fixed-parameter algorithm.

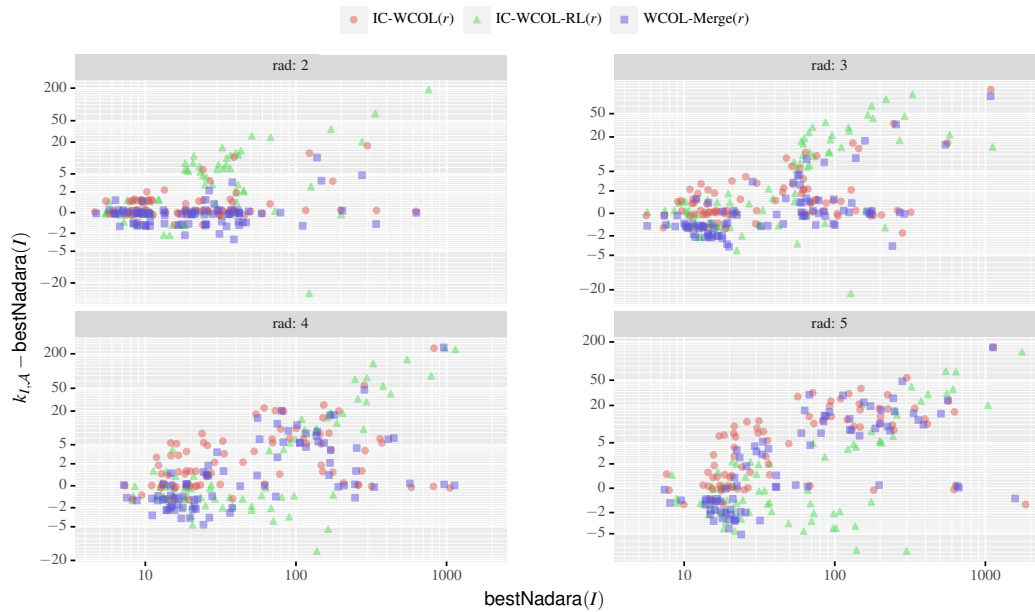
Table 3 contains the results for IC-WCOL-RL(r). Although the relative and absolute improvements of turbocharging the Degree-Heuristic are slightly higher, the quality ratios for the turbocharged version of the Sreach-Heuristic are significantly better. This could imply that IC-WCOL-RL(r) struggles to turbocharge slightly worse heuristics such as the Degree-Heuristic. Furthermore, we see that for the Sreach-Heuristic the quality ratios are better for larger radii. The reason for this could be that the Sreach-Heuristic performs well for larger radii even before turbocharging. We also notice that for the medium graph class the quality ratios get worse. The reason is that the implementation of IC-WCOL-RL(r) is slightly more computationally expensive than for the other approaches.

■ **Table 3** IC-WCOL-RL(r): White columns give relative improvements, light gray columns give quality ratios, dark gray columns give average/maximum absolute improvements. For improvements, we compare to the underlying heuristic without turbocharging and without local search. Time limit: 300s.

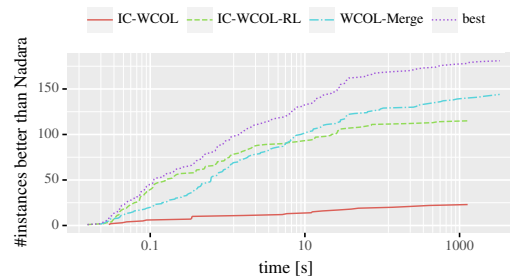
tests	r	Sreach IC-WCOL-RL(r)			Degree IC-WCOL-RL(r)		
small	2	-2.1%	3.2/28	15.9%	-7.5%	3.1/11	22.0%
	3	1.6%	2.7/10		-7.4%	4.4/19	
	4	3.6%	2.5/7		-7.9%	5.2/21	
	5	3.3%	3.2/8		-8.9%	6.3/26	
medium	2	-8.8%	3.1/12	10.7%	-14.8%	3.0/8	15.5%
	3	-3.8%	3.8/16		-12.6%	5.7/25	
	4	-0.6%	3.8/14		-16.4%	5.8/17	
	5	1.4%	4.3/11		-18.5%	7.6/22	
big	2	-14.8%	3.0/10	6.4%	-29.8%	4.2/12	10.1%
	3	-13.4%	7.1/26		-25.8%	8.8/23	
	4	-7.7%	7.4/19		-28.2%	13.1/60	
	5	-3.1%	6.5/19		-28.2%	12.2/31	
huge	2	-22.8%	14.4/47	5.4%	-26.3%	22.1/76	7.1%
	3	-16.2%	11.5/21		-29.6%	12.4/20	
	4	-17.2%	5.5/13		-27.4%	8.2/16	
	5	2.0%	4.3/12		-14.4%	4.0/10	

Achieved coloring numbers in comparison to Nadara et al. Figure 2 illustrates the distribution of results for all turbocharging algorithms in a scatter plot. Data points below y -value zero mean that the turbocharging algorithm improves a bound on the weak r -coloring number of an instance. Among our approaches, we see that while WCOL-MERGE(r) performs well for small radii, IC-WCOL-RL(r) performs well for larger radii. Together with the analysis from above we can conclude that Nadara et al.’s approaches yield lower weak coloring numbers on average but there is fraction of roughly one half instances where our approaches supersede Nadara et al.’s, in particular if the computed weak coloring numbers are small. Overall, we improved bounds for 172 of the 334 considered instances after $t = 300s$. The resulting new bounds are lower by 5% on average over all instances. Follow-up investigations also showed that the relative improvement of turbocharging negatively correlates with the average degree of the graph, suggesting that our turbocharging algorithms work better for particularly sparse graphs.

As mentioned, our approach has the advantage that investing more time yields gradually better results. Figure 3 shows the number of instances for which a turbocharging approach improve weak coloring numbers compared to Nadara et al. after a specific time. That is, for a time t , the y -value of a line corresponds to the number of instances I such that $k_{I,A,H,t} < \text{bestNadara}(I)$. The values for the line $best$ are determined by taking the number of instances I where any of the turbocharging approaches is better than $\text{bestNadara}(I)$. Interestingly, IC-WCOL-RL(r) starts off with more improved instances, however, after 3600 seconds, WCOL-MERGE(r) achieved more instances with smaller coloring numbers than Nadara et al. After 3600 seconds, IC-WCOL(r) was able to improve 24 instances compared to Nadara et al., WCOL-MERGE(r) 144 instances, and IC-WCOL-RL(r) 115 instances. Overall, we could improve upper bounds for 181 of the 334 instances with $t = 3600s$. These are nine more than for $t = 300s$.



■ **Figure 2** Scatter plot of the results for turbocharging algorithms broken down by radius. Each data point corresponds to an instance and a turbocharging algorithm. The x -value is the best weak coloring number achieved by Nadara et al. for this instance, the y -value is the difference of coloring numbers between the best weak coloring number achieved by Nadara et al. and the weak coloring number $k_{I,\mathcal{A}}$ achieved by the turbocharging algorithm \mathcal{A} (minimum over both turbocharged heuristics). The x -axis is scaled logarithmically and the y -axis is scaled pseudo-logarithmically. Time limit: 300 s.



■ **Figure 3** Line plot of the number of instances where a turbocharging approach achieves better coloring numbers than Nadara et al. over time. The time is scaled logarithmically.

5 Conclusion

On the theoretical side, we determined obstructions (running-time lower bounds) and promising avenues (a fixed-parameter algorithm) for applying the turbocharging framework to computing vertex orderings of small weak coloring numbers. On the experimental side, on a diverse set of instances each of the turbocharging approaches we use yields large improvements over the plain heuristics. This is most pronounced for the fixed-parameter turbocharging WCOL-MERGE(r). Then we compared turbocharging to the best results gained by the seven heuristics that Nadara et al. [32] employed together with local search

procedures. Turbocharging so far yields on average larger weak coloring numbers than the best of the previous approaches. However, for 173 of the in total 334 instances, turbocharging outperforms all of the previous approaches combined. It works particularly well for small computed coloring numbers and sparse input instances.

References

- 1 Faisal N. Abu-Khzam, Shaowei Cai, Judith Egan, Peter Shaw, and Kai Wang. Turbo-charging dominating set with an FPT subroutine: Further improvements and experimental analysis. In T. V. Gopal, Gerhard Jäger, and Silvia Steila, editors, *Proceedings of the 14th Annual Conference on Theory and Applications of Models of Computation (TAMC 2017)*, volume 10185 of *Lecture Notes in Computer Science*, pages 59–70, 2017. doi:10.1007/978-3-319-55911-7_5.
- 2 Saeed Akhoondian Amiri, Patrice Ossona de Mendez, Roman Rabinovich, and Sebastian Siebertz. Distributed domination on graph classes of bounded expansion. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA 2018)*, pages 143–151. ACM, 2018. doi:10.1145/3210377.3210383.
- 3 Alex Pothén. The complexity of optimal elimination trees, 1988.
- 4 Hans L. Bodlaender, Jitender S. Deogun, Klaus. Jansen, Ton. Kloks, Dieter. Kratsch, Haiko. Müller, and Zsolt. Tuza. Rankings of graphs. *SIAM Journal on Discrete Mathematics*, 11(1):168–181, 1998. doi:10.1137/S0895480195282550.
- 5 Michael Breen-McKay, Brian Lavalée, and Blair D. Sullivan. Hardness of the generalized coloring numbers. *CoRR*, abs/2112.10562, 2021. arXiv:2112.10562.
- 6 Marco Bressan. Faster algorithms for counting subgraphs in sparse graphs. *Algorithmica*, 83(8):2578–2605, 2021. doi:10.1007/s00453-021-00811-0.
- 7 Marcin Briański, Piotr Micek, Michał Pilipczuk, and Michał T. Seweryn. Erdős–hajnal properties for powers of sparse graphs. *SIAM Journal on Discrete Mathematics*, 35(1):447–464, 2021. doi:10.1137/20M1342756.
- 8 Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In *Proceedings of the Second International Workshop on Parameterized and Exact Computation (IWPEC 2006)*, pages 239–250. Springer, 2006. doi:10.1007/11847250_22.
- 9 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>.
- 10 Erik D. Demaine, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, Somnath Sikdar, and Blair D. Sullivan. Structural sparsity of complex networks: Bounded expansion in random models and real-world graphs. *Journal of Computer and System Sciences*, 105:199–241, 2019. doi:10.1016/j.jcss.2019.05.004.
- 11 Alexander Dobler. Turbocharging heuristics for weak coloring numbers. Master’s thesis, TU Wien, 2021. doi:10.34726/hss.2021.91580.
- 12 Alexander Dobler. Turbocharging heuristics for weak coloring numbers: Source code, November 2021. doi:10.5281/zenodo.5732923.
- 13 Alexander Dobler, Manuel Sorge, and Anaïs Villedieu. Turbocharging heuristics for weak coloring numbers. *CoRR*, abs/2203.03358, 2022. arXiv:2203.03358.
- 14 R. G. Downey, J. Egan, M. R. Fellows, F. A. Rosamond, and P. Shaw. Dynamic dominating set and turbo-charging greedy heuristics. *Tsinghua Science and Technology*, 19(4):329–337, 2014. doi:10.1109/TST.2014.6867515.
- 15 Jan Dreier. Lacon- and shrub-decompositions: A new characterization of first-order transductions of bounded expansion classes. In *Proceedings of the 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021)*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470680.

- 16 Zdeněk Dvořák. Constant-factor approximation of the domination number in sparse graphs. *European Journal of Combinatorics*, 34(5):833–840, 2013. doi:10.1016/j.ejc.2012.12.004.
- 17 Kord Eickmeyer, Archontia C. Giannopoulou, Stephan Kreutzer, O.-joungh Kwon, Michał Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Neighborhood complexity and kernelization for nowhere dense classes of graphs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 63:1–63:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.63.
- 18 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics*, 18:3.1:3.1–3.1:3.21, 2013. doi:10.1145/2543629.
- 19 Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julián Mestre, and Stefan Rümmele. Turbocharging treewidth heuristics. *Algorithmica*, 81(2):439–475, 2019. doi:10.1007/s00453-018-0499-1.
- 20 Martin Grohe, Stephan Kreutzer, Roman Rabinovich, Sebastian Siebertz, and Konstantinos S. Stavropoulos. Coloring and covering nowhere dense graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2467–2481, 2018. doi:10.1137/18M1168753.
- 21 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *Journal of the ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.
- 22 Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the bron-kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):35, 2017. doi:10.1007/s13278-017-0455-0.
- 23 Falk Hüffner, Christian Komusiewicz, and Manuel Sorge. Finding highly connected subgraphs. In Giuseppe F. Italiano, Tiziana Margaria-Steffen, Jaroslav Pokorný, Jean-Jacques Quisquater, and Roger Wattenhofer, editors, *SOFSEM 2015: Theory and Practice of Computer Science*, pages 254–265. Springer, 2015. doi:10.1007/978-3-662-46078-8_21.
- 24 Gwenaël Joret, Piotr Micek, Patrice Ossona de Mendez, and Veit Wiechert. Nowhere dense graph classes and dimension. *Combinatorica*, 39(5):1055–1079, 2019. doi:10.1007/s00493-019-3892-8.
- 25 Meir Katchalski, William McCuaig, and Suzanne Seager. Ordered colourings. *Discrete Mathematics*, 142(1):141–154, 1995. doi:10.1016/0012-365X(93)E0216-Q.
- 26 Hal A. Kierstead and Daqing Yang. Orderings on graphs and game coloring number. *Order*, 20(3):255–264, 2003. doi:10.1023/B:ORDE.0000026489.93166.cb.
- 27 Christian Komusiewicz and Manuel Sorge. An algorithmic framework for fixed-cardinality optimization in sparse graphs applied to dense subgraph problems. *Discrete Applied Mathematics*, 193:145–161, 2015. doi:10.1016/j.dam.2015.04.029.
- 28 Łukasz Kowalik, Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge, and Piotr Wygocki. The pace 2020 parameterized algorithms and computational experiments challenge: Treedepth. In Yixin Cao and Marcin Pilipczuk, editors, *Proceedings of the 15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.37.
- 29 O-joungh Kwon, Michał Pilipczuk, and Sebastian Siebertz. On low rank-width colorings. *European Journal of Combinatorics*, 83:103002, 2020. doi:10.1016/j.ejc.2019.103002.
- 30 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983. doi:10.1145/2402.322385.
- 31 R. Milošević, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.
- 32 Wojciech Nadara, Marcin Pilipczuk, Roman Rabinovich, Felix Reidl, and Sebastian Siebertz. Empirical evaluation of approximation algorithms for generalized graph coloring and uniform quasi-wideness. *ACM Journal of Experimental Algorithmics*, 24(1):2.6:1–2.6:34, 2019. doi:10.1145/3368630.

- 33 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Springer, 2012.
- 34 Jaroslav Nešetřil, Patrice Ossona de Mendez, Michał Pilipczuk, and Xuding Zhu. Clustering powers of sparse graphs. *The Electronic Journal of Combinatorics*, pages P4.17–P4.17, 2020. doi:10.37236/9417.
- 35 Michael P. O’Brien and Blair D. Sullivan. An experimental evaluation of a bounded expansion algorithmic pipeline. *arXiv:1712.06690 [cs]*, 2017. arXiv:1712.06690.
- 36 Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Parameterized circuit complexity of model-checking on sparse structures. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 789–798. ACM, 2018. doi:10.1145/3209108.3209136.
- 37 Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web (WWW 2017)*, pages 1431–1440. International World Wide Web Conferences Steering Committee, 2017. doi:10.1145/3038912.3052597.
- 38 N. Przulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, 2004. doi:10.1093/bioinformatics/bth436.
- 39 Felix Reidl and Blair D. Sullivan. A color-avoiding approach to subgraph counting in bounded expansion classes. *arXiv:2001.05236 [cs]*, 2020. arXiv:2001.05236.
- 40 Felix Reidl, Fernando Sánchez Villaamil, and Konstantinos Stavropoulos. Characterising bounded expansion by neighbourhood complexity. *European Journal of Combinatorics*, 75:152–168, 2019. doi:10.1016/j.ejc.2018.08.001.
- 41 Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494:86–98, 2013. doi:10.1016/j.tcs.2012.12.049.
- 42 Sebastian Siebertz. Nowhere dense graph classes and algorithmic applications. a tutorial at highlights of logic, games and automata 2019. *arXiv:1909.06752 [cs]*, 2019. arXiv:1909.06752.

A Local Search Algorithm for Large Maximum Weight Independent Set Problems

Yuanyuan Dong¹ ✉ 

Dallas, TX, USA

Andrew V. Goldberg ✉

Amazon.com, Santa Clara, CA, USA

Alexander Noe ✉ 

Amazon.com, Bellevue, WA, USA

Nikos Parotsidis¹ ✉

Department of Computer Science, University of Copenhagen, Denmark

Mauricio G.C. Resende ✉  

Amazon.com, Bellevue, WA, USA

Industrial & Systems Engineering, University of Washington, Seattle, WA, USA

Quico Spaen ✉ 

Amazon.com, Santa Clara, CA, USA

Abstract

Motivated by a real-world vehicle routing application, we consider the maximum-weight independent set problem: Given a node-weighted graph, find a set of independent (mutually nonadjacent) nodes whose node-weight sum is maximum. Some of the graphs arising in the vehicle routing application are large, having hundreds of thousands of nodes and hundreds of millions of edges.

To solve instances of this size, we develop a new local search algorithm, which is a metaheuristic based on the greedy randomized adaptive search (GRASP) framework. This algorithm, named METAMIS, uses a wider range of simple local search operations than previously described in the literature. We introduce data structures that make these operations efficient. A new variant of path-relinking is introduced to escape local optima and so is a new alternating augmenting-path local search move that improves algorithm performance.

We compare an implementation of our algorithm with a state-of-the-art publicly available code on public benchmark sets, including some large instances. Our algorithm is, in general, competitive and outperforms this openly available code on large vehicle routing instances of the maximum weight independent set problem. We hope that our results will lead to even better maximum-weight independent set algorithms.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases GRASP, local search, maximum-weight independent set, path-relinking, heuristic, metaheuristic

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.45

Related Version *Full Version*: <https://arxiv.org/abs/2203.15805>

1 Introduction

Given an undirected graph $G = (V, E)$, where V is the set of nodes and E the set of edges, an *independent set* $S \subseteq V$ is a set of mutually non-adjacent nodes of graph G . If each node $v \in V$ is assigned a weight w_v , a maximum-weight independent set (*MWIS*) of nodes $S^* \subseteq V$ is an independent set whose sum of weights, $W(S^*) = \sum_{v \in S^*} w_v$ is maximum. We denote $n = |V|$ and $m = |E|$.

¹ The work was done while the author was at Amazon.com



A simple way to state MWIS is as an *Integer Linear Program (ILP)*. Let x_v be a binary decision variable such that $x_v = 1$ if node $v \in S \subseteq V$ and $x_v = 0$ otherwise, where S is an independent set of nodes. A simple integer programming (IP) formulation for selecting a maximum-weight independent set of nodes is

$$\begin{aligned} & \max \sum_{v \in V} w_v x_v \\ & \text{subject to} \\ & \quad x_u + x_v \leq 1, \forall (u, v) \in E \\ & \quad x_v \in \{0, 1\}, \forall v \in V. \end{aligned}$$

A well-known way to strengthen the formulation is to add *clique inequalities*. Let C be a subset of all cliques in the input graph. We add the constraints

$$\sum_{v \in Q} x_v \leq 1 \quad \forall Q \in C.$$

While these constraints are redundant for the ILP problem, they strengthen the linear programming relaxation of the problem.

MWIS is a classical optimization problem that has been extensively studied and has many applications [2]. Solving the MWIS problem is hard. It is one of Karp's original NP-complete problems [10, 12]. The problem is also hard to approximate [11]. Over the years, heuristics have been the workhorse for solving large instances of the maximum independent set problem approximately [18]. In particular, the most successful heuristics have been the ones based on metaheuristic algorithms, such as GRASP [8], tabu search [9], and iterated local search [1, 17].

In this paper we introduce *METAMIS*, a new metaheuristic algorithm for the MWIS problem. METAMIS is based on the *greedy randomized adaptive search procedure* – GRASP [20], with *truncated path-relinking* [19]. GRASP is a procedure consisting of iterations made up from successive constructions of a greedy randomized solution and subsequent iterative improvements of it through a local search, and path-relinking is a technique for escaping local optima by generating intermediate solutions along a path that connects two known high-quality solutions. Our motivation is a long-haul vehicle routing (VR) application that yields large MWIS problems, some with close to 900 thousand nodes. Compared to benchmark instances used in previously published work, the VR-MWIS instances are often larger and have a very different structure. We conduct experiments with METAMIS on MWIS instances arising in different applications, including on our VR-MWIS instances and on other publicly available ones. Due to page limit, we omit some of the details of our implementation. See the full version of the paper [4] for details.

We start with known local search moves and perturbation techniques and introduce new local search moves with data structures to make these moves efficient. We also introduce improved perturbation technique variants. Although our algorithm is a general-purpose heuristic, our motivation comes from the VR problem. A variant of our algorithm takes advantage of the application-specific features. In this application, we have a good initial solution which can be used to for warm-start. In addition, graphs from this application come with a large set of known cliques. This allows us to get a good relaxed LP solution, which we use to guide local search.

Due to the page limit, we omit some of the algorithm and implementation details and focus only on the benchmark from our motivating vehicle routing application. We also omit some intuition and discussions. The full paper [4] covers this material.

2 High Level Description

The MWIS algorithm is an iterative local search algorithm based on the *Greedy Randomized Adaptive Search Procedure (GRASP)* metaheuristic, which is a general metaheuristic for combinatorial optimization [6, 7, 20]. The algorithm also uses *path-relinking* to escape local optima [15, 20].

Algorithm 1 Algorithm Overview.

```

1: procedure MWIS( $G = (V, E, w)$ , maxTime,  $S_0$ )
2:    $S \leftarrow \text{localSearch}(G, S_0)$ 
3:    $\mathcal{ES} \leftarrow \{\}$  ▷ Empty set of elite solutions
4:    $\mathcal{ES}.\text{add}(S)$ 
5:   while  $t \leq \text{maxTime}$  do
6:      $S_G \leftarrow \text{findRandomizedGreedySolution}(G)$ 
7:     if LsBeforeRelinking then ▷ Optional local search
8:        $S_G \leftarrow \text{localSearch}(G, S_G)$ 
9:     end if
10:     $S_e \leftarrow \mathcal{ES}.\text{randomEliteSolution}()$ 
11:     $S' \leftarrow \text{pathRelinking}(G, S_G, S_e)$ 
12:     $S' \leftarrow \text{localSearch}(G, S')$ 
13:     $\mathcal{ES}.\text{tryToAddAndEvict}(S')$  ▷ Add solution to elite set, if full evict similar
    solution of lesser value (or don't insert if no worse elite solution exists)
14:  end while
15:  return  $\mathcal{ES}.\text{bestSolution}()$ 
16: end procedure

```

Algorithm 1 gives a high-level view of the algorithm. In addition to the graph, the input to the algorithm includes a stopping criterion, e.g., a time limit, and an initial solution. When no such solution is available, one can find a solution using the randomized greedy algorithm described later in this section. The algorithm applies local search to improve the initial solution and enters the main loop. At termination of the local search procedure, we are at a local optimum.

The algorithm maintains a set of *elite* solutions \mathcal{ES} , which are the best solutions we have seen so far. We add a solution to \mathcal{ES} immediately after a local search, so the elite solutions are always locally optimal. At each iteration of the loop, we first attempt to escape the local optimum corresponding to the elite solution. In this process, we can decrease the objective function. To escape a local optimum, we first find a randomized greedy solution S_G . Optionally, we apply local search to improve S_G . Then we apply path-relinking to S_G and a random elite solution from \mathcal{ES} to find a new solution S' . Then we apply local search to improve S' , and update S^* if we find a better solution.

For the VR-MWIS instances, the algorithm variant without the optional local search (on line 8) works better, so we omit the search for these instances. We also set the size of the elite set \mathcal{ES} to 1, so we only retain the best solution. This setting works best for the VR-MWIS instances. For other problem families, different parameter choices were found to work better [13, 14].

2.1 Greedy Algorithm

The GRASP framework needs a randomized greedy procedure that produces diverse initial solutions.

2.2 Local Search

■ Algorithm 2 Local Search Procedure

```

1: procedure LOCALSEARCH( $G = (V, E, w), S, \text{numIterations}$ )
2:    $i \leftarrow 1$ 
3:    $S^* \leftarrow S$ 
4:   while  $i \leq \text{numIterations}$  do
5:      $S_i \leftarrow \{\}$  ▷ Empty solution
6:     while  $w(S_i) < w(S)$  do ▷ Repeat until no improvement is found
7:        $S_i \leftarrow S$ 
8:        $S \leftarrow \text{starOneMoves}(G, S)$ 
9:        $S \leftarrow \text{AAPMoves}(G, S)$ 
10:       $S \leftarrow \text{oneStarMoves}(G, S)$ 
11:      if  $w(S_i) < w(S)$  then break ▷ Solution improved
12:      end if
13:       $S \leftarrow \text{twoStarMoves}(G, S)$ 
14:    end while
15:    if  $w(S) > w(S^*)$  then
16:       $S^* \leftarrow S$ 
17:       $i \leftarrow 1$ 
18:    else
19:       $S \leftarrow \text{perturb}(S)$ 
20:    end if
21:  end while
22:  return  $S^*$ 
23: end procedure

```

The local search procedure, outlined in Algorithm 2, repeatedly performs local moves with positive gain. We aim to find positive gain (*improving*) moves until we reach a local optimum, and then we perform a random perturbation of the solution. If we find an improving move, we apply it immediately. We use a subset of (x, y) moves and *alternating augmenting path moves* (AAP-moves). While the (x, y) moves have been studied previously, the AAP moves are new. We describe the moves at a high level in this section, and give a detailed description in Section 3.

An (x, y) move removes x nodes from the solution and adds y nodes to it while maintaining solution independence. We use $*$ instead of x or y to denote an arbitrary positive integer. Note that the number of applicable moves increases significantly as x and y increase. Previous algorithms used (x, y) moves for small values of x and y . In particular, the algorithm of [17] uses $(*, 1)$ and $(1, *)$ moves. Our algorithm uses $(*, 1)$, $(1, *)$, and $(2, *)$ moves. The number of $(2, *)$ moves is large. We use data structures and operation ordering that make improving moves more likely, which makes our algorithm more efficient. If an (x, y) move renders S non-maximal, we add nodes without a neighbor in S to the independent set in random order until S becomes maximal. Note that through this update sequence, S remains an independent set.

A $(*, 1)$ move inserts a single node u into the current solution S and removes its neighbors from S . Procedure $\text{starOneMoves}(G, S)$ applies the $(*, 1)$ moves until there is no such improving move.

A $(1, *)$ move removes a node v from S and adds to S an independent subset I of the nodes whose only neighbor in S before the removal is v . Usually one has multiple choices of independent sets to add. A good heuristic is to add a maximum weight set of the neighbors that maintains independence. This is done when the number of neighbors is small (at most seven in our experiments). We use a naive recursive algorithm: Pick a node u in the neighborhood and recursively solve two subproblems. The first subproblem results by adding u to S and deleting its neighbors from the graph. We get the second subproblem by deleting u without adding it to S . The better of the two corresponding solutions is returned. Procedure $\text{oneStarMoves}(G, S)$ applies the $(1, *)$ moves until there is no such improving move.

A $(2, *)$ move removes two nodes, u and v , from S and adds to S an independent subset I of the nodes whose only neighbors in S before the removal is u , or v , or both u and v . Generally, this set is significantly larger than the corresponding set for the $(1, *)$ moves, and the recursive operation used for the $(1, *)$ moves is too expensive. One could use greedy addition, but in our experiments a random addition, that adds to S a random node from I that has no neighbors in S , was better. Procedure $\text{twoStarMoves}(G, S)$ applies the $(2, *)$ moves until it finds an improving move or there is no improving $(2, *)$ move. Note that unlike the corresponding procedures for other moves, twoStarMoves exits as soon as it finds an improving move.

Our idea for AAP moves comes from matching algorithms [5], although we use a somewhat different definition. Given an independent set S , we define an AAP P as follows. Let $I = S \cap P$ and $O = P - S$ be nodes of P that are in and out of S , respectively.

1. if $v \in I$, then the neighbors of v on P are in O ,
2. if $v \in O$, then the neighbors of v on P are in I ,
3. if we *flip* the path, i.e., set $S = S - I + O$, S remains an independent set.

An AAP move finds an alternating augmenting path, flips it, and looks at the change in $w(S)$. If the change is positive, we accept the AAP move; otherwise we reject the move. For efficiency, we apply a limited number of AAP moves. Procedure $\text{AAPMoves}(G, S)$ applies the AAP moves until there is no such improving move or we reach the limit on the number of AAP moves.

During an execution of the algorithm, most local search moves do not improve solution quality and thus do not change the solution. Note that complexity of evaluating $(2, *)$ moves is significantly higher than those for the other moves. Our local search repeatedly applies starOneMoves , AAPMoves , and oneStarMoves procedures while these procedures find improving moves. If we find an improving move, an immediate application of these procedures may find additional improvements due to neighborhood changes, so we iterate. Only when these procedures fail to find improving moves we call twoStarMoves . If twoStarMoves fails to improve the solution, we perform a random perturbation.

The perturbation adds a small set of random nodes to S and removes their neighbors. After perturbing, we resume local search. The local search algorithm terminates if there has been no improvement to the best solution after a predefined number of iterations.

2.3 Using the Relaxed LP Solution

In our VR application, we use clique information and get a relaxed LP solution to the relaxed problem. The solution assigns a value $x_v \in [0, 1]$ to each node v . We use these values to bias random node selection in the perturbation step of the local search. When performing

a random perturbation in Algorithm 2, we add a node v to the solution with probability proportional to $x_v + \epsilon$. Here ϵ is a positive value (set to $\epsilon = 0.005$) that ensures that each node can be picked, even if $x_v = 0$. This guides the local search by biasing route selection toward nodes with higher fractional relaxed solution value. Using prefix sums we can pick a random node in time $\mathcal{O}(\log |V|)$: We draw a random floating-point number $z \in [0, \sum_{v \in V} x_v)$ and use binary search on the prefix sum array to pick a node such that the sum up to but excluding the node is less than z , and the sum up to and including the node is greater or equal to z .

2.4 Adaptive Path-relinking

Path-relinking is a technique for escaping local optima by generating intermediate solutions along a path that connects two known high-quality solutions. We discuss this technique in the context of MWIS and reversible local search moves. Define an undirected graph associated with the search space MWIS, where the nodes correspond to feasible solutions and the edges correspond to local search moves that transform the solution corresponding to the tail of the edge to the solution corresponding to the head. A path in this graph corresponds to a sequence of the moves that transform the solution at one end of the path into a solution at the other end. Note that the moves need not improve the objective function value. The underlying assumption of path-relinking is that if the end-points of a path correspond to high quality solutions, then the path will contain previously undiscovered high-quality solutions.

For our local search, given two solutions S and T , we can transform S into T as follows. Initialize $S' = S$. At every step, we do either a $(*, 1)$ move or a $(1, *)$ move. In the former case, pick a node $v \in T - S'$, add v to S' , and remove neighbors of v from S' . In the latter case, pick a node $v \in S'$, $v \notin T$ and remove v from S' . Let $N(v)$ denote the set of neighbors of v . Then we iterate over nodes u in $N(v) \cap T$. If $N(u) \cap S' = \emptyset$, we add u to S' .

For large graphs, finding good solutions is expensive. Instead of combining two good solutions, we apply path-relinking to combine the randomized greedy solution S_G with the current best solution S^* , which is locally optimal. While S^* is a good solution, S_G may not be good, and the solutions on the path far from S^* are usually not good either. We modify path-relinking so that it examines only a prefix of the path close to S^* . The prefix is small enough so that the solution quality remains good, yet big enough so that the subsequent local search will not end up with a locally optimal solution equivalent to S^* . This an adaptive variant of the *truncated greedy path-relinking* described in [19].

The first modification is to choose the node x to add to S or to remove from S greedily. We pick a node that maximizes the weight of the solution we get after the move. A second modification is to do a truncated path-relinking: we stop the process after a certain number of steps, which we adjust adaptively. We start with a small limit on the number of steps and increase the limit if the algorithm gets stuck in a local optimum of weight $w(S^*)$.

3 Data Structures and Optimizations

For large graphs, the choice of data structures is important for the efficiency of the algorithm. When making trade-offs between performance on sparse and dense graphs we favor the former because our motivating application yields relatively sparse graphs.

Several of our data structures use sets of objects. We use a representation of sets based on hashing. This representation allows constant time addition, deletion, and membership query, and linear time iteration over all set elements. We also assume that if we add an element to the set that already contains the element, the set does not change. Similarly, if we delete an element not in the set, the set does not change.

3.1 Input Graph

The input graph is static: it does not change throughout the execution. We assign to the nodes of the graph integer IDs from $[0, \dots, n - 1]$ and place them in an array, with node i in position i . Each node has an array of edges incident to it. This places the edges incident to a node in contiguous memory locations, assuring that a common operation of scanning an edge list has a good memory locality. We sort edges by IDs of the head node. This allows us to do neighborhood queries (e.g., “Is v in $N(u)$?”) in time logarithmic in the degree of u using binary search.

Note that using sets to represent neighborhoods would give constant neighborhood queries and linear time edge list scan. However, the constant factors, both in terms of running time and memory consumption, associated with hashing are large. In addition, we lose the locality in edge list scans. For graphs arising from our motivating application, the array-based implementation is significantly faster than the one based on sets.

3.2 Interstate Graph

The *interstate graph* makes the local search operations more efficient. To describe this graph, we need a few definitions.

For a node $u \in S$, $(u, v) \in E$, we say that v is a *1-tight neighbor* of u if $N(v) \cap S = \{u\}$ [1]. Note that if we remove u from S , we can add to S any 1-tight neighbor of u .

Two nodes $u, v \in S$ are *mates* if for at least one node $w \notin S$, w has exactly two neighbors in S : $N(w) \cap S = \{u, v\}$. We call the node w a *2-tight neighbor* of u and v . We say that w is a 2-tight neighbor of u if u has a mate v such that w is a 2-tight neighbor of u and v . If we delete u and v from S , we can replace them by an independent set of the union of three sets: the set of the 1-tight neighbors of u , the set of 1-tight neighbors of v , and the set of the shared 2-tight neighbors of u and v .

Our main data structure is the *interstate graph* $G_{IS} = (V, E_{IS}, w)$. For G_{IS} , the nodes and node weights are the same as in the input graph G . The edge set E_{IS} is changed dynamically depending on the nodes in the current independent set S . E_{IS} has three types of edges:

1. $e = (u, v) \in E$, where $u \in S$ and v is a *1-tight neighbor* of u ;
2. $e = (u, w) \in E$, where $u \in S$ and w is a *2-tight neighbor* of u ;
3. $e = (u, v)$, where $u, v \in S$ are *mates*.

We represent the three edge types separately.

1. For every $u \in S$, we represent its 1-tight neighbors as sets. For $v \notin S$ that is a 1-tight neighbor of u we add the *1-tight edge* (v, u) .
2. For every pair of mates u and v , we maintain a set of 2-tight neighbors of u and v . For every 2-tight neighbor $w \notin S$, we add the pair of *2-tight edges* (w, u) and (w, v) .
3. For every node v in S , we maintain a set M_v of its mates. Every mate $w \in M_v$ corresponds to a *mate edge* (v, w) .

3.3 Efficient Implementation of (x, y) Moves

In this section we show how to efficiently implement (x, y) moves using the interstate graph and two additional optimizations, one for the $(1, *)$ moves and another for $(*, y)$ moves. We discuss maintenance of the interstate graph in Section 3.2.

To implement $(*, 1)$ operations efficiently, we use an idea from [17]. For every $u \notin S$, we maintain a value

$$\Delta(u) = w(u) - \sum_{v \in S \cap N(u)} w(v)$$

to speed up the $(*, 1)$ moves. Such a move is an improving move when $\Delta(u) > 0$. We keep a set S^+ of the nodes u with $\Delta(u) > 0$. Note that for an efficient implementation of $(*, 1)$ moves, we need to update the vector $\Delta(\cdot)$ and the set S^+ . We do this as follows. Every time we add a node u to S , we remove u from S^+ . Then for each $v \in N(u)$, $v \notin S$, we set $\Delta(v) = \Delta(v) - w(u)$. Every time we remove u from S , we scan the edge list of u and compute $\Delta(u)$. If $\Delta(u) > 0$, we add u to S^+ . Also during the scan, for every neighbor v of u such that $v \notin S$, we increase $\Delta(v)$ by $w(u)$, and if $\Delta(v)$ becomes positive, we add v to S^+ . We have an improving $(*, 1)$ move if and only if S^+ is non-empty. In this case, we can pick a node u from S^+ and apply the $(*, 1)$ move to it.

Since for every $u \in S$ we maintain a set of its 1-tight neighbors as a hash set, we can efficiently run the recursive or the greedy algorithm described in Section 2 on this set. Similarly, since for every $u \in S$ we maintain the set of its mates, we can iterate over all mates of u . Furthermore, for a pair of mates u and v , we have the set of the common 2-tight neighbors, and we can apply the randomized algorithm to this set.

Next we describe an optimization that prunes $(1, *)$ and $(2, *)$ moves that are unlikely to improve the solution. For the $(1, *)$ move that removes u , we evaluate the move only if the 1-tight neighborhood of u changed since the last time we evaluated the move but failed to improve the solution. We say that the neighborhood changed if we add u to S and u has a non-trivial 1-tight neighborhood. Since our implementation of the $(1, *)$ move is deterministic and depends only on the 1-tight neighborhood, we know that the move will fail. We maintain the set S_1 of nodes $u \in S$ whose 1-tight neighborhood changed but is not empty. We pick nodes for $(1, *)$ moves from S_1 . While initializing G_{IN} , we initialize S_1 to include all nodes with non-trivial 1-tight neighborhoods. When we update G_{IN} , we also update S_1 (see Section 3.5).

For the $(2, *)$ move, we maintain a set S_2 of mate pairs $\{u, v\}$ which are eligible for the move. We delete a pair from S_2 and evaluate the move that removes this pair from S . We add a pair $\{u, v\}$ to S_2 when they become 2-tight mates, or when $\{u, v\}$ are 2-tight mates and their 2-tight neighborhood changes, or when they are 2-tight mates and the 1-tight neighborhood of either u or v changes. Our implementation of the $(2, *)$ move depends only on the 2-tight neighborhood of the mates. However, the implementation is randomized. Although it is possible that one evaluation of the move succeeds and another fails when the 2-tight neighborhood stays the same, we assume this is unlikely and prune the move. We maintain the set S_2 of mates whose 2-tight neighborhood changed. We pick mates for $(2, *)$ moves from S_2 . While initializing G_{IN} , we initialize S_2 to all pairs of mates. When we update G_{IN} , we update S_2 as well.

3.4 AAP Moves

For efficiency, we only look for alternating augmenting paths (AAPs) in the interstate graph. The only edges on any AAP are either edges from members of S to their 1-tight and 2-tight neighbors (as edges between 2-tight mates would not yield an alternating path). To limit the number of AAP move evaluations, we start a search for an AAP from a 1-tight neighbor of $v \in S_1$ (S_1 was introduced in Section 3.3). This way we guarantee that the move will not decrease the cardinality of S , making the move more likely to succeed. The alternating path initially contains v and its single neighbor $u \in S$. We grow the path as follows. Let $u \in S$ be

the last node on the current AAP, and let U be the set of nodes on the AAP that are in S and \bar{U} be the set of nodes on the AAP that are not in S . We pick a mate w and a 2-tight neighbor x of u such that

- x is not a neighbor of any node of \bar{U} in the input graph (so that the extended path will be an AAP),
- neither x nor w are already in AAP,
- the gain of flipping the extended path is maximized.

If we succeed in finding such a $\{w, x\}$ pair, we add w and x to the path. Then we redefine u to be x and continue growing the path. To introduce additional randomness, we increase the gain for every $\{w, x\}$ pair by a random real number $\epsilon \in [-\delta, \delta]$ and maximize the perturbed gains. We use $\delta = 50$ in our experiments. We terminate the search if the length of the path exceeds a threshold or the gain of flipping the path falls below a (negative) threshold. We then perform the highest positive gain move that flips a prefix of the final path. If no positive gain move is encountered, we do nothing (the move fails).

3.5 Maintaining the Interstate Graph

The vast majority of the local search moves we evaluate do not improve the solution and G_{IN} does not change. We need to update the graph only when a move succeeds, which happens rarely. Our data structures speed up move evaluations and support move pruning. The added overhead is in data structure initialization and updates. The update complexity is non-trivial, but for sparse graphs the complexity is much smaller than the time we save due to the improved move efficiency and pruning.

Let $\rho(u) = |N(u) \cap S|$ denote the number of the neighbors of u in S . Note that for nodes $u \in S$, $\rho(u) = 0$. We maintain $\rho(u)$ for all nodes $u \in V$.

Given an initial solution S , we build G_{IN} , S_1 , and S_2 as follows. We process all nodes $u \notin S$. For each u , we scan its edge list in G and initialize $\rho(u)$. If $\rho(u) = 1$, we let $N(u) \cap S = \{v\}$, add the 1-tight edge (u, v) to the edge list of u in G_{IN} , and add u to the set of 1-tight neighbors of v . If $\rho(u) = 2$, we let $N(u) \cap S = \{v, w\}$, add v to the set of mates of w and add w to the set of mates of v . We also add the pair of 2-tight edges (u, v) and (u, w) to G_{IN} . Finally, we add u to the set of 2-tight neighbors of the mates $\{v, w\}$. We initialize S_1 to the set of all nodes $u \in S$ with non-empty set of 1-tight neighbors. We initialize S_2 to the set of all mate pairs $\{u, v\}$. The initialization takes linear time.

Our algorithm updates S by removing a set of nodes S^- and adding a set of nodes S^+ . We break the update into a sequence of single-node updates: first we remove nodes of S^- one by one, then we add nodes of S^+ one by one. We update G_{IN} after each individual update of S .

After removing a node u from S , we empty its set of 1-tight neighbors and remove u from S_1 . For each mate v of u , we set the corresponding set of 2-tight neighbors to empty and remove u from the set of mates of v . We also remove the pair $\{u, v\}$ from S_2 . Afterwards, we empty the set of mates of u . We then visit its neighbors $v \in V \setminus S$. For each neighbor v , we decrement $\rho(v)$. We need to update G_{IN} if $\rho(v)$ becomes zero, one, or two.

Cases for zero and two are simpler. If the value is zero, we set the 1-tight neighbor of v to null. If the value is two, let $N(v) \cap S = \{a, b\}$. We can find a and b by scanning the edge list of v in G . We add a to the set of mates of b and vice versa. We also add v to the set of 2-tight neighbors of $\{a, b\}$. Finally, we add the 2-tight pair of edges (v, a) and (v, b) to G_{IN} .

If the value is one, we have to update both the old 2-tight neighborhood and the new 1-tight neighborhood. For the latter, we set the 1-tight neighbor of v to the unique neighbor $w \in S$, and add v to the 1-tight neighbor set of w . For the former update, note that v was a 2-tight neighbor for mates $\{w, x\}$ for some $w \in S$ before the removal of v . We remove v from the set of 2-tight neighbors of w and delete the 2-tight edge pair (v, w) and (v, x) from G_{IN} .

Now consider the addition of a node u to S that maintains the independence of S . We scan the edge list of u and for all neighbors v (guaranteed not to be in S) and increment $\rho(v)$. We need to update G_{IN} if $\rho(v)$ becomes one, two, or three.

Cases for one and three are simpler. If the value is one, we add the 1-tight edge (v, u) to G_{IN} , add v to the set of 1-tight neighbors of u , and add u to S_1 . If the value is three, v has a pair of 2-tight edges (v, a) and (v, b) , where a and b are mates. We delete (v, a) and (v, b) from G_{IN} . Then we remove v from the set of 2-tight neighbors of a and b . If the set becomes empty, a and b are no longer mates, so we remove a from the set of mates of b , remove b from the list of mates of a , and remove $\{a, b\}$ from S_2 .

If the value is two, we have to update both the old 1-tight neighborhood and the new 2-tight neighborhood. For the former, let (v, w) be the 1-tight edge. We remove the edge and remove v from the set of 1-tight neighbors of w . If the set becomes empty, we remove w from S_1 . In the latter case, $N(v) \cap S = \{v, w\}$ for some $w \in S$. We add u to the set of mates of w and vice versa. We also add v to the set of 2-tight neighbors of v and w . Finally, we add $\{a, b\}$ to S_2 .

Note that since when we add or remove u to or from S , we may need to scan edge lists of multiple neighbors of u , updating G_{IN} when G is dense may be expensive.

4 Experimental results

4.1 Algorithms and Computational Environment

We implemented our algorithm, which we call *METAMIS*, in Java because it is used in a production system at Amazon and Java is a requirement. For the same reason, we use doubles for node weights. Furthermore, due to licensing restrictions, we use only standard Java libraries. We compiled our code using Java 8.

Although one can tune our algorithm for specific problem families, we use fixed parameter settings in all experiments.

We compare our implementation to the *ILSVND* algorithm of [17]. The publicly available code of [17] is implemented in C++ and represents weights using integers. We made one modification to ILSVND: added the ability to warm start from an initial solution. Given a solution in the input, we initialize the current solution of ILSVND to the input solution. We compiled ILSVND using full optimization (-O3).

For a given instance, algorithm time-quality plots give a lot of information about relative performance of the algorithms. For example, one algorithm may dominate another, or one can converge to a better solution but take longer to converge, etc. The algorithms we compare are stochastic and algorithm performance depends on the pseudo-random seed we use. Furthermore, the algorithms we compare do not know if and when they reach an optimal solution. Usually there is a chance that a solution may improve. However, the algorithms *converge* in a sense that it may reach a point of diminishing returns when a substantial improvement becomes unlikely. To compare the two algorithms, we put a time limit T on their executions. For different problem families, the limit may be different. We run each instance with five different pseudo-random seeds and report the best solution value the algorithm finds. In many cases the algorithms converge. However, for harder problems this may take too long, and the algorithms do not converge within the time limit.

For representative instances, we give the time-quality plots, but we have too many instances to give all the plots. Therefore, we report solution quality at times $T/10$ and $T/2$. In addition, we report the time t^* defined as follows. For a given problem instance, consider the set of final solution values over all algorithms and seed values. Let s be the smallest one

of these values. For a given algorithm, consider the run producing the best final solution value. For this algorithm, we define t^* to be the earliest time this run reaches the value of s or higher. Intuitively, we are comparing best runs of the algorithms being evaluated.

For graph algorithms, C++ is usually faster than Java by a factor from three to six. We expect this to hold for our algorithm as well, especially since we make heavy use of standard Java hash set library, which incurs significant overhead compared to C++. Although we do not adjust the runtimes we report, one has to keep this in mind that if re-implemented in C++, our algorithm would be faster.

We run our experiments on an AWS r3.4xlarge instance with 122GiB RAM and 16 virtual CPUs on Intel Xeon Ivy Bridge processors.

4.2 Computational Results

Our full study [4] uses three benchmark families, but due to the page limit we focus on the benchmark from our motivating application, vehicle routing [3]. In this application, the MWIS problem comes up in several contexts, and we have several instances for each of these contexts.

Tables and plots appear in the appendix. Table 1 lists the *VR instances* with their sizes. The number of nodes in these instances ranges from 979 to 883,238; the number of edges ranges from 3,140 to 389,304,424. The instances are moderately sparse, but the density tends to grow with the problem size. The average degree is below 4 on some small instances and over 400 on some large ones.

Table 1 has additional information: values for the initial solutions we use and upper bounds on optimal solution values. We obtain the upper bounds by solving the corresponding LP relaxation problems to optimality. The initial solution are good: their values are close to the upper bound. Note that an optimal solution may not achieve the upper bound.

For VR instances, we have additional information: relaxed LP solutions and initial solutions. We use this information in practice as it yields better results. In our experiments, we give results both for runs with and runs without initial solutions. We also run our algorithm with initial solutions but without the relaxed solutions to see how much a good initial solution matters, and to have an apples to apples comparison with ILSVND, which does not use this information.

In this section we discuss *VR Instances* [3], which motivated our work. Plot for 2-hour runs of all algorithms on one of the largest instances, CR-S-L-4, given in Figure 1, provides insight into relative algorithm performance. All codes converge, and METAMIS dominates corresponding ILSVND runs. Without warm start, ILSVND solution is worse than the initial solution while METAMIS finds a better solution. With warm start, both algorithms find better solutions. Although plots for METAMIS with and without LP data look very close. However, Table 3 shows that the best solution value with LP was 1% better than without LP: 5,775,704 vs. 5,715,256.

Next we discuss performance of VR instances in detail. Here we set the time limit $T = 3600$ seconds. Table 2 gives results for MWIS with no additional data. For each instance in the table, column $w_{10\%}$ shows the best solution value found at time point $T/10$, column $w_{50\%}$ shows the best solution value found at time point $T/2$, and column w shows the best solution value found when the process is finished at time T . METAMIS finds better solutions than ILSVND except for three instances. For two instances, MT-D-01 and MT-W-01, solution quality is the same. On MW-W-01, the ILSVND solution is better, but only by 0.8%. All three exceptions happen on smaller instances and both algorithms converge quickly. There is no improvement after time $T/10$.

An interesting observation is that on MT-D-01 and MT-W-01, solution values match the corresponding upper bounds given in Table 1, so the solutions are optimal. Since the upper bound need not be tight, it is possible that we solve other instances to optimality, but do not have a proof.

On larger instances, METAMIS has better final values as well as better values at times $T/10$ and $T/2$. On the problem with the highest number of nodes, CR-S-L-3, the difference in the final values is 2.1%. Note that on large instances, neither algorithms converged in time T .

Table 3 shows results for the VR instances for METAMIS+LP, METAMIS, and ILSVND. Note that on three instances, MT-D-FN, MW-D-FN, and MW-W-FN, ILSVND fails to improve the initial solution and t^* is undefined. METAMIS improves the solution on these instances, probably due to a more sophisticated set of local search operations. While both algorithms allow a warm start from a given solution, the METAMIS+LP version of our algorithm uses clique information to compute the relaxed LP solution, and uses it to guide local search. We evaluate both versions of METAMIS to see how much the LP relaxation helps. As in the case of no initial solution, the algorithms converge on most of the small instances and do not converge on larger instances.

Recall that with no initial solution, we found optimal solutions for MT-D-01 and MT-W-01. With the initial solution, METAMIS+LP finds an optimal solution for two more instances, MT-W-FN and MR-W-FN. METAMIS finds an optimal solution for the latter instance, but not for the former. ILSVND does not find any new optimal solutions.

Next we discuss the effect of a good initial solution, comparing results for METAMIS and ILSVND from Tables 2 and 3. Comparing initial solution values from Table 1 with solutions obtained by solving the problems from scratch, we see that in many cases, the initial solution is better than the solution computed from scratch. In fact, for ILSVND, most solutions are worse than the corresponding initial solution. This confirms that our initial solutions are good.

With the warm start, both variants of our algorithm, METAMIS and METAMIS+LP, dominate ILSVND, producing same or (in most cases) better quality solutions. ILSVND is also slower on all instances except one.

To evaluate the benefit of using LP relaxation, we compare METAMIS+LP with METAMIS. On most instances, METAMIS+LP dominates METAMIS. The latter never finds a better solution. For about 1/3 of the instances, solution quality is the same, and for the remaining 2/3, METAMIS+LP performs better. The same holds for intermediate times $T/10$ and $T/2$ except for one instance at $T/2$ where METAMIS solution value is slightly better.

5 Concluding remarks

We developed METAMIS for a real-world VR application for which even a small improvement in solution quality yields substantial cost reduction. Our study is the first to include the benchmark of VR instances [3]. We show that METAMIS works well on the VR instances. We also observed that the VR instances have a structure that is different from that of computer, road, and social network (CRS) instances [16]. The main result of Lamm [16] are local transformations which reduce a MWIS problem to an equivalent problem that is much smaller. On the VR instances, the transformations failed to reduce the problem size significantly.

Our full paper shows that METAMIS works well of the CRS instances. The algorithm of Lamm [16] solves these instances to optimality. Instances of Lamm [16] are hard to reproduce due to weight randomization. In the full paper, we define weights so that they are easy to reproduce. It would be interesting to run the algorithm of Lamm [16] and compare the results.

METAMIS uses a more sophisticated set of local search moves and introduces data structures and lazy evaluation techniques that facilitate efficient implementation of these moves. We also introduce a new variation of path-relinking tailored to large problems. In addition, we show how to use a good relaxed solution to guide local search. These techniques add to the metaheuristic design toolset. We hope that our ideas will lead to even more efficient MWIS algorithms. The ideas may also prove useful in metaheuristic algorithms for other problems.

References

- 1 D.V. Andrade, M.G.C. Resende, and R.F. Werneck. Fast local search for the maximum independent set problem. *J. of Heuristics*, 18:525–547, 2012.
- 2 S. Butenko. *Maximum independent set and related problems with applications*. PhD thesis, U. of Florida, Gainesville, Florida, 2003.
- 3 Y. Dong, A.V. Goldberg, A. Noe, N. Parotsidis, M.G.C. Resende, and Q. Spaen. New instances for maximum weight independent set from a vehicle routing application. *Operations Research Forum*, 2(48), 2021. doi:10.1007/s43069-021-00084-x.
- 4 Y. Dong, A.V. Goldberg, A. Noe, N. Parotsidis, M.G.C. Resende, and Q. Spaen. A Metaheuristic Algorithm for Large Maximum Weight Independent Set Problems. Technical Report arXiv:2203.15805, arXiv.org, 2022.
- 5 J. Edmonds. Paths, trees, and flowers. *Can. J. Math.*, 17:449–467, 1965.
- 6 T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.
- 7 T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
- 8 T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.
- 9 C. Friden, A. Hertz, and D. de Werra. STABULUS: A technique for finding stable sets in large graphs with tabu search. *Computing*, 42:35–44, 1989.
- 10 M.R. Garey and D.S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Company, San Francisco, 1979.
- 11 J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.
- 12 R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972.
- 13 A.F. Kummer. Personal communication, 2020.
- 14 A.F. Kummer, M.G.C. Resende, and M. Souto. Automatic algorithm configuration and selection of MetaMIS for maximum independent set. Technical report, Amazon MMPROS, Seattle, 2020.
- 15 M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.
- 16 S. Lamm, C. Schulz, D. Strash, R. Williger, and Huashuo Zhang. Exactly solving the maximum weight independent set problem on large real-world graphs. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019*, pages 144–158. SIAM, 2019. doi:10.1137/1.9781611975499.12.
- 17 B. Nogueira, R.G.S. Pinheiro, and A. Subramanian. A hybrid iterated local search heuristic for the maximum weight independent set problem. *Optimization Letters*, 12(3):567–583, 2018.

18 M. Pelillo. Heuristics for maximum clique and independent set. In C.A. Floudas and P.M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1508–1520. Springer US, Boston, MA, 2009.

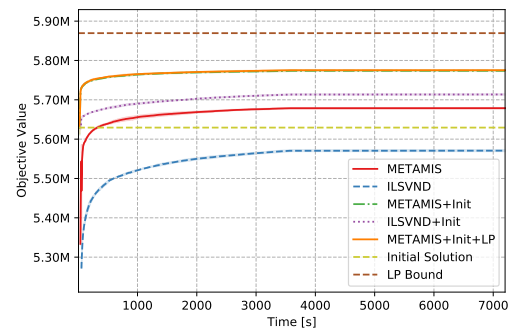
19 M.G.C. Resende, R. Martí, M. Gallego, and A. Duarte. GRASP and path relinking for the max-min diversity problem. *Computers & Operations Research*, 37:498–508, 2010.

20 M.G.C. Resende and C.C. Ribeiro. *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer, New York, 2016.

A Appendix: Tables and Plots

■ **Table 1** VR instances.

Graph	V	E	Initial Sol.	LP bound
MT-D-01	979	3 841	228 874 404	238 166 485
MT-D-200	10 880	547 529	286 750 411	287 228 467
MT-D-FN	10 880	645 026	290 723 959	290 881 566
MT-W-01	1 006	3 140	299 132 358	312 121 568
MT-W-200	12 320	554 288	383 620 215	384 099 118
MT-W-FN	12 320	593 328	390 596 383	390 869 891
MW-D-01	3 988	19 522	465 730 126	477 563 775
MW-D-20	10 790	718 152	522 485 254	531 510 712
MW-D-40	33 563	2 169 909	533 938 531	543 396 252
MW-D-FN	47 504	4 577 834	542 182 073	549 872 520
MW-W-01	3 079	48 386	1 268 370 807	1 270 311 626
MW-W-05	10 790	789 733	1 328 552 109	1 334 413 294
MW-W-10	18 023	2 257 068	1 342 415 152	1 360 791 627
MW-W-FN	22 316	3 495 108	1 350 675 180	1 373 020 454
MR-D-01	14 058	60 738	1 664 446 852	1 695 332 636
MR-D-03	21 499	168 504	1 739 544 141	1 763 685 757
MR-D-05	27 621	295 700	1 775 123 794	1 796 703 313
MR-D-FN	30 467	367 408	1 794 070 793	1 809 854 459
MR-W-FN	15 639	267 908	5 386 472 651	5 386 842 781
CW-T-C-1	266 403	162 263 516	1 298 968	1 353 493
CW-T-C-2	194 413	125 379 039	933 792	957 291
CW-T-D-4	83 091	43 680 759	457 715	463 672
CW-T-D-6	83 758	44 702 150	457 605	463 946
CW-S-L-1	411 950	316 124 758	1 622 723	1 677 563
CW-S-L-2	443 404	350 841 894	1 692 255	1 759 158
CW-S-L-4	430 379	340 297 828	1 709 043	1 778 589
CW-S-L-6	267 698	191 469 063	1 159 946	1 192 899
CW-S-L-7	127 871	89 873 520	589 723	599 271
CR-T-C-1	602 472	216 862 225	4 605 156	4 801 515
CR-T-C-2	652 497	240 045 639	4 844 852	5 032 895
CR-T-D-4	651 861	245 316 530	4 789 561	4 977 981
CR-T-D-6	381 380	128 658 070	2 953 177	3 056 284
CR-T-D-7	163 809	49 945 719	1 451 562	1 469 259
CR-S-L-1	863 368	368 431 905	5 548 904	5 768 579
CR-S-L-2	880 974	380 666 488	5 617 351	5 867 579
CR-S-L-4	881 910	383 405 545	5 629 351	5 869 439
CR-S-L-6	578 244	245 739 404	3 841 538	3 990 563
CR-S-L-7	270 067	108 503 583	1 969 254	2 041 822



■ **Figure 1** Time-quality plot for CR-S-L-4. Note that the plots for METAMIS+Init and METAMIS+Init+LP are very close.

■ **Table 2** Results on VR instances with no additional information.

Name	METAMIS				ILSVND			
	$w_{10\%}$	$w_{50\%}$	w	t^* [s]	$w_{10\%}$	$w_{50\%}$	w	t^* [s]
MT-D-01	238 166 485	238 166 485	238 166 485	0.948	238 166 485	238 166 485	238 166 485	1.290
MT-D-200	286 976 422	287 048 909	287 048 909	188.1	286 838 210	286 838 210	286 943 799	2.276
MT-D-FN	290 866 943	290 866 943	290 866 943	104.4	290 393 532	290 666 380	290 666 380	561.6
MT-W-01	312 121 568	312 121 568	312 121 568	0.278	312 121 568	312 121 568	312 121 568	0.080
MT-W-200	383 818 136	383 961 099	383 961 323	1.433	383 865 836	383 896 403	383 896 403	1.036
MT-W-FN	390 688 944	390 830 057	390 854 593	568.1	390 715 890	390 798 842	390 798 842	709.2
MW-D-01	476 099 262	476 164 209	476 334 711	267.9	475 653 439	475 906 790	475 906 790	1.173
MW-D-20	524 255 389	525 036 493	525 124 575	85.40	520 854 115	522 415 092	523 138 978	2.685
MW-D-40	533 934 442	535 707 479	536 520 199	81.36	530 227 261	532 272 896	532 400 878	1.830
MW-D-FN	539 754 400	541 372 345	541 918 916	98.34	532 663 872	537 238 784	537 674 129	2.466
MW-W-01	1 270 305 952	1 270 305 952	1 270 305 952	0.500	1 246 949 460	1 246 949 460	1 246 949 460	23.66
MW-W-05	1 328 958 047	1 328 958 047	1 328 958 047	19.96	1 327 687 399	1 328 707 787	1 328 707 787	984.8
MW-W-10	1 340 878 388	1 342 899 725	1 342 899 725	1.204	1 331 002 512	1 341 482 310	1 342 067 985	1.876
MW-W-FN	1 349 369 736	1 350 818 543	1 350 818 543	527.7	1 334 835 589	1 348 128 240	1 350 159 705	3.584
MR-D-01	1 689 074 331	1 689 520 690	1 689 781 114	15.52	1 683 529 331	1 686 091 786	1 687 842 856	2.906
MR-D-03	1 753 188 475	1 753 968 167	1 754 110 286	20.34	1 743 429 914	1 747 269 072	1 749 972 580	3.257
MR-D-05	1 784 519 403	1 785 664 042	1 786 342 921	19.56	1 770 832 093	1 774 407 092	1 777 876 780	3.595
MR-D-FN	1 795 912 642	1 797 284 091	1 797 573 192	22.65	1 779 897 201	1 785 545 729	1 788 331 878	3.388
MR-W-FN	5 357 026 363	5 358 386 615	5 358 386 615	1.442	5 352 347 338	5 370 471 580	5 371 649 721	461.6
CW-T-C-1	1 310 223	1 315 122	1 317 775	94.52	1 290 974	1 299 279	1 302 478	3.585
CW-T-C-2	924 664	929 626	931 802	189.7	914 736	921 021	922 858	3.599
CW-T-C-4	454 769	456 565	457 185	324.4	452 035	453 741	454 544	2.365
CW-T-D-6	455 823	457 382	457 790	70.48	452 366	454 254	454 254	1.582
CW-S-L-1	1 623 280	1 630 417	1 634 950	261.9	1 603 051	1 615 247	1 620 756	3.597
CW-S-L-2	1 695 131	1 704 424	1 708 820	225.3	1 670 836	1 685 870	1 690 536	3.596
CW-S-L-4	1 712 553	1 722 542	1 725 591	173.7	1 689 318	1 701 309	1 706 264	3.599
CW-S-L-6	1 150 229	1 156 916	1 158 925	138.4	1 136 356	1 142 720	1 145 694	3.086
CW-S-L-7	582 925	585 929	587 288	125.2	577 087	581 583	581 583	1.278
CR-T-C-1	4 617 204	4 644 635	4 654 419	58.16	4 508 901	4 558 780	4 576 695	3.598
CR-T-C-2	4 834 040	4 863 054	4 874 346	62.29	4 715 023	4 772 847	4 789 909	3.600
CR-T-D-4	4 778 868	4 808 490	4 817 281	56.91	4 663 588	4 716 258	4 734 674	3.598
CR-T-D-6	2 945 721	2 964 007	2 970 011	94.09	2 896 260	2 921 540	2 929 671	3.574
CR-T-D-7	1 431 915	1 438 896	1 440 281	148.4	1 411 061	1 423 279	1 426 400	3.581
CR-S-L-1	5 547 038	5 575 602	5 588 489	72.42	5 400 658	5 464 532	5 487 254	3.595
CR-S-L-2	5 652 928	5 680 688	5 691 892	57.91	5 491 814	5 561 766	5 586 973	3.580
CR-S-L-4	5 634 886	5 671 369	5 681 336	65.09	5 477 340	5 550 943	5 572 856	3.573
CR-S-L-6	3 833 391	3 851 432	3 859 513	92.45	3 751 019	3 793 995	3 808 314	3.599
CR-S-L-7	1 977 161	1 986 354	1 989 879	90.90	1 940 573	1 957 872	1 963 579	3.584

Table 3 METAMIS+LP, METAMIS, ILSVND results on VR instances with start solution.

Name	METAMIS+LP			METAMIS			ILSVND					
	$w_{10\%}$	$w_{50\%}$	w	$t^* [s]$	$w_{10\%}$	$w_{50\%}$	w	$t^* [s]$	$w_{10\%}$	$w_{50\%}$	w	
MT-D-01	238 166 485	238 166 485	238 166 485	0.109	238 166 485	238 166 485	238 166 485	0.373	238 166 485	238 166 485	238 166 485	1.473
MT-D-20	287 038 328	287 048 081	287 048 081	69.51	287 010 847	287 018 324	287 036 715	122.6	286 949 274	286 973 561	286 973 561	363.1
MT-D-FN	290 771 450	290 771 450	290 771 450	—	290 752 054	290 771 450	290 771 450	—	290 723 959	290 723 959	290 723 959	—
MT-W-01	312 121 568	312 121 568	312 121 568	0.122	312 121 568	312 121 568	312 121 568	0.320	312 121 568	312 121 568	312 121 568	0.063
MT-W-200	383 971 124	383 985 408	383 985 408	893.0	383 804 298	383 986 483	383 986 483	1.343	383 808 376	383 979 962	383 979 962	1.721
MT-W-FN	390 828 160	390 869 891	390 869 891	139.9	390 787 880	390 848 998	390 805 960	710.1	390 805 960	390 805 960	390 805 960	196.2
MW-D-01	475 886 356	475 987 082	475 987 082	270.2	475 549 969	475 814 986	475 955 989	2.278	475 523 699	475 732 519	475 825 497	2.134
MW-D-20	525 052 532	525 402 318	525 486 034	8.694	524 574 519	525 068 939	525 192 291	7.699	523 248 884	523 248 884	523 248 884	26.98
MW-D-40	535 705 687	536 210 247	536 735 155	0.434	535 436 892	535 711 417	536 092 070	0.474	534 040 009	534 040 009	534 040 009	7.797
MW-D-FN	543 098 071	543 622 238	543 857 187	—	542 740 347	543 253 226	543 374 394	—	542 182 073	542 182 073	542 182 073	—
MW-W-01	1 269 314 742	1 269 344 846	1 269 344 846	672.0	1 269 344 846	1 269 344 846	1 269 344 846	0.603	1 269 344 846	1 269 344 846	1 269 344 846	1.247
MW-W-05	1 328 958 047	1 328 958 047	1 328 958 047	0.431	1 328 958 047	1 328 958 047	1 328 958 047	0.447	1 328 955 871	1 328 955 871	1 328 955 871	4.266
MW-W-10	1 342 915 691	1 342 915 691	1 342 915 691	0.511	1 342 915 691	1 342 915 691	1 342 915 691	1.255	1 342 847 887	1 342 847 887	1 342 847 887	19.39
MW-W-FN	1 350 814 699	1 350 814 699	1 350 818 543	—	1 350 771 010	1 350 818 542	1 350 818 543	—	1 350 675 180	1 350 675 180	1 350 675 180	—
MR-D-01	1 688 024 106	1 688 777 944	1 689 278 470	7.245	1 687 486 503	1 687 807 619	1 688 118 984	16.84	1 684 211 854	1 686 046 636	1 686 452 467	2.763
MR-D-03	1 756 186 736	1 756 989 875	1 757 227 519	5.123	1 755 768 835	1 756 154 528	1 756 337 669	12.31	1 751 006 933	1 752 345 436	1 752 769 459	3.305
MR-D-05	1 787 220 357	1 787 666 207	1 787 849 777	19.91	1 786 084 687	1 786 734 327	1 786 755 817	73.22	1 782 046 226	1 782 560 957	1 783 836 981	3.525
MR-D-FN	1 798 215 807	1 798 926 794	1 799 452 160	17.40	1 798 075 911	1 798 571 155	1 798 661 823	38.60	1 794 949 819	1 794 949 819	1 796 037 791	3.564
MR-W-FN	5 386 842 781	5 386 842 781	5 386 842 781	0.503	5 386 842 781	5 386 842 781	5 386 842 781	0.855	5 386 838 669	5 386 838 669	5 386 838 669	10.01
CW-T-C-1	1 334 884	1 336 953	1 338 064	30.69	1 333 129	1 335 297	1 336 563	22.44	1 322 410	1 326 551	1 327 556	3.501
CW-T-C-2	944 404	945 748	945 886	25.86	943 366	944 785	945 565	27.72	939 568	940 356	940 356	701.3
CW-T-D-4	460 643	461 027	461 056	2.000	460 554	460 852	461 025	1.828	458 360	458 360	458 360	48.65
CW-T-D-6	460 982	461 223	461 312	2.717	460 815	461 057	461 174	2.706	459 096	459 096	459 096	80.73
CW-S-L-1	1 656 404	1 660 475	1 660 815	46.27	1 656 404	1 660 475	1 660 815	90.11	1 644 241	1 649 006	1 651 483	3.585
CW-S-L-2	1 731 077	1 735 964	1 738 128	85.11	1 730 208	1 734 736	1 736 245	109.3	1 714 923	1 722 672	1 724 930	3.452
CW-S-L-4	1 748 029	1 752 354	1 753 803	91.08	1 746 941	1 751 474	1 751 988	84.05	1 733 007	1 739 992	1 742 459	3.553
CW-S-L-6	1 174 005	1 175 931	1 177 156	27.48	1 174 169	1 175 886	1 176 233	33.79	1 167 611	1 169 914	1 170 096	1.886
CW-S-L-7	593 045	593 744	593 891	4.825	593 077	593 744	593 891	6.622	591 398	591 398	591 398	161.2
CR-T-C-1	4 730 533	4 739 684	4 743 040	17.92	4 725 855	4 735 644	4 738 289	18.10	4 665 849	4 687 422	4 696 568	3.591
CR-T-C-2	4 954 613	4 966 121	4 968 952	25.80	4 950 818	4 962 045	4 964 446	19.83	4 891 697	4 912 140	4 920 058	3.585
CR-T-D-4	4 898 377	4 908 285	4 911 646	19.69	4 896 504	4 906 792	4 909 999	17.86	4 836 312	4 859 311	4 867 272	3.597
CR-T-D-6	3 017 902	3 022 448	3 024 523	28.67	3 016 890	3 022 046	3 023 349	40.35	2 990 174	2 999 852	3 004 067	3.593
CR-T-D-7	1 458 949	1 459 958	1 460 240	16.88	1 458 571	1 459 653	1 459 948	8.115	1 455 226	1 456 048	1 456 048	752.0
CR-S-L-1	5 672 398	5 686 939	5 692 891	36.79	5 668 764	5 685 884	5 690 515	21.79	5 590 089	5 617 916	5 630 437	3.596
CR-S-L-2	5 763 866	5 780 859	5 784 034	24.90	5 759 512	5 775 002	5 780 449	22.53	5 670 522	5 701 371	5 715 430	3.589
CR-S-L-4	5 756 016	5 771 410	5 777 081	24.08	5 755 282	5 771 391	5 775 704	24.18	5 676 163	5 701 271	5 715 256	3.598
CR-S-L-6	3 926 517	3 933 476	3 936 137	22.24	3 923 574	3 932 059	3 935 089	19.00	3 883 092	3 898 898	3 905 831	3.597
CR-S-L-7	2 014 584	2 018 371	2 019 428	34.09	2 013 466	2 017 034	2 017 836	40.24	1 998 320	2 006 129	2 007 794	3.488

SAT Backdoors: Depth Beats Size

Jan Dreier  

Algorithms and Complexity Group, TU Wien, Austria

Sebastian Ordyniak  

Algorithms and Complexity Group, University of Leeds, UK

Stefan Szeider  

Algorithms and Complexity Group, TU Wien, Austria

Abstract

For several decades, much effort has been put into identifying classes of CNF formulas whose satisfiability can be decided in polynomial time. Classic results are the linear-time tractability of Horn formulas (Aspvall, Plass, and Tarjan, 1979) and Krom (i.e., 2CNF) formulas (Dowling and Gallier, 1984). Backdoors, introduced by Williams, Gomes and Selman (2003), gradually extend such a tractable class to all formulas of bounded distance to the class. Backdoor size provides a natural but rather crude distance measure between a formula and a tractable class. Backdoor depth, introduced by Mählmann, Siebertz, and Vigny (2021), is a more refined distance measure, which admits the utilization of different backdoor variables in parallel. Bounded backdoor size implies bounded backdoor depth, but there are formulas of constant backdoor depth and arbitrarily large backdoor size.

We propose FPT approximation algorithms to compute backdoor depth into the classes Horn and Krom. This leads to a linear-time algorithm for deciding the satisfiability of formulas of bounded backdoor depth into these classes. We base our FPT approximation algorithm on a sophisticated notion of obstructions, extending Mählmann et al.’s obstruction trees in various ways, including the addition of separator obstructions. We develop the algorithm through a new game-theoretic framework that simplifies the reasoning about backdoors.

Finally, we show that bounded backdoor depth captures tractable classes of CNF formulas not captured by any known method.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases satisfiability, backdoor (depth)

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.46

Related Version *Full Version*: <https://arxiv.org/abs/2202.08326>

Funding *Sebastian Ordyniak*: Supported by the Engineering and Physical Sciences Research Council (EPSRC, project EP/V00252X/1).

Stefan Szeider: supported by the Austrian Science Fund (FWF, project P32441) and the Vienna Science and Technology Fund (WWTF, project ICT19-065).

1 Introduction

Deciding the satisfiability of a propositional formula in conjunctive normal form (CNFSAT) is one of the most important NP-complete problems [4, 16]. Despite its theoretical intractability, heuristic algorithms work surprisingly fast on real-world CNFSAT instances [7]. A common explanation for this discrepancy between theoretical hardness and practical feasibility is the presence of a certain “hidden structure” in realistic CNFSAT instances [14]. There are various approaches to capturing the vague notion of a “hidden structure” with a mathematical concept. One widely studied approach is to consider the hidden structure in terms of decomposability. For instance, CNFSAT can be solved in quadratic time for classes of CNF formulas of bounded branchwidth [2] or bounded treewidth [24].



© Jan Dreier, Sebastian Ordyniak, and Stefan Szeider;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 46; pp. 46:1–46:18



Leibniz International Proceedings in Informatics

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

A complementary approach proposed by Williams et al. [26] considers the hidden structure of a CNFSAT instance in terms of a small number of key variables, called *backdoor variables*, that when instantiated move the instance into a polynomial-time solvable class. More precisely, a *backdoor*¹ of size k of a CNF formula F into a polynomial-time solvable class \mathcal{C} is a set B of k variables such that for all partial assignments τ to B , the instantiated formula $F[\tau]$ belongs to \mathcal{C} . In fact, CNFSAT can be solved in linear time for any class of CNF formulas that admit backdoors of *bounded size* into the class of *Horn*, *dual Horn* or *Krom* (i.e., 2CNF) formulas².

The size of a smallest backdoor of a CNF formula F into a class \mathcal{C} is a fundamental but rather simple distance measure between F and \mathcal{C} . Mählmann, Siebertz, and Vigny [17] proposed to consider instead the smallest *depth* over all backdoors of a formula F into a class \mathcal{C} as distance measure. It is recursively defined as follows:

$$\text{depth}_{\mathcal{C}}(F) := \begin{cases} 0 & \text{if } F \in \mathcal{C} \\ 1 + \min_{x \in \text{var}(F)} \max_{\epsilon \in \{0,1\}} \text{depth}_{\mathcal{C}}(F[x = \epsilon]) & \text{if } F \notin \mathcal{C} \text{ and } F \text{ is connected} \\ \max_{F' \in \text{Conn}(F)} \text{depth}_{\mathcal{C}}(F') & \text{otherwise} \end{cases} \quad (1)$$

$\text{Conn}(F)$ denotes the set of connected components of F ; precise definitions are given in Section 2. We can certify $\text{depth}_{\mathcal{C}}(F) \leq k$ with a *component \mathcal{C} -backdoor tree* of depth $\leq k$ which is a decision tree that reflects the choices made in the above recursive definition.

Backdoor depth is based on the observation that if an instance F decomposes into multiple connected components of $F[x = 0]$ and $F[x = 1]$, then each component can be treated independently. This way, one is allowed to use in total an unbounded number of backdoor variables. However, as long as the depth of the component \mathcal{C} -backdoor tree is bounded, one can still utilize the backdoor variables to solve the instance efficiently. In the context of graphs, similar ideas are used in the study of tree-depth [19, 20] and elimination distance [3, 6]. Bounded backdoor size implies bounded backdoor depth, but there are classes of formulas of unbounded backdoor size but bounded backdoor depth.

The challenging algorithmic problem \mathcal{C} -BACKDOOR DEPTH is to find for a fixed base class \mathcal{C} and a given formula F , a component \mathcal{C} -backdoor tree of F of depth $\leq k$. Mählmann et al. [17] gave an FPT-approximation algorithm for this problem, with k as the parameter) where \mathcal{C} is the trivial class NULL for formulas without variables. A component NULL-backdoor tree must instantiate all variables of F .

New Results. In this paper, we give the first positive algorithmic results for backdoor depth into nontrivial classes. A minimization problem admits a *standard fixed-parameter tractable approximation (FPT-approximation)* [18] if for an instance of size n and parameter k there is an *FPT-algorithm*, i.e., an algorithm running in time $f(k)n^{\mathcal{O}(1)}$, that either outputs a solution of size at most $g(k)$ or outputs that the instance has no solution of size at most k , for some computable functions f and g ; $g(k)$ is also referred to as the performance ratio of the algorithm.

► **Main Result 1** (Theorem 15). *\mathcal{C} -BACKDOOR DEPTH admits an FPT-approximation algorithm if \mathcal{C} is any of the Schaefer classes Horn, dual Horn, or Krom.*

¹ We focus only on strong backdoors, as weak backdoors only apply to satisfiable formulas.

² According to Schaefer's Theorem [25], these three classes are the largest nontrivial classes of CNF formulas defined in terms of a property of clauses, for which CNFSAT can be solved in polynomial time.

Since our FPT algorithms have linear running time for fixed backdoor depth k , we obtain the following corollary:

► **Main Result 2** (Corollary 16). *CNFSAT can be solved in linear time for formulas of bounded backdoor depth into the Schaefer classes Horn, dual Horn, and Krom.*

Backdoor depth is a powerful parameter that is able to capture and exploit structure in CNFSAT instances that is not captured by any other known method. We list some well-known parameters which render CNFSAT fixed-parameter tractable (the list is not complete but covers some of the most essential parameters). For all these parameters, there exist CNF formulas with constant backdoor depth (into *Horn*, *dual Horn*, and *Krom*) but where the other parameter is arbitrarily large: (i) backdoor size into Horn, dual Horn, and Krom [21]; (ii) number of leaves of backdoor trees into Horn, dual Horn, and Krom [23, 22]; (iii) backdoor depth into the class of variable-free formulas [17]; (iv) backdoor treewidth to Horn, dual Horn, and Krom [9, 8]; (v) backdoor size into heterogeneous base classes based on Horn, dual Horn, and Krom [11]; (vi) backdoor size into scattered base classes based on Horn, dual Horn, and Krom [10]; (vii) deletion backdoor size into the class of quadratic Horn formulas [12]; (viii) backdoor size into bounded incidence treewidth [13]. We give definitions and separation proofs in the full version.

Approach and Techniques. A common approach to construct backdoors is to compute in parallel both an upper bound and a lower bound. The upper bounds are obtained by constructing the backdoor itself, and lower bounds are usually obtained in the form of so-called *obstructions*. These are parts of an instance that are proven to be “far away” from the base class. Our results and techniques build upon the pioneering work by Mählmann et al. [17], who introduce *obstruction trees* for backdoor depth. A main drawback of their approach is that it is limited to the trivial base class NULL, where the obstructions are rather simple because they can contain only boundedly many variables. Our central technical contribution is overcoming this limitation by introducing *separator obstructions*.

Separator obstructions allow us to algorithmically work with obstruction trees containing an unbounded number of variables, an apparent requirement for dealing with nontrivial base classes different from NULL. In the context of backdoor depth, it is crucial that an existing obstruction is disjoint from all potential future obstructions, so they can later be joined safely into a new obstruction of increased depth. Mählmann et al. [17] ensure this by placing the whole current obstruction tree into the backdoor – an approach that only works for the most trivial base class because only there the obstructions have a bounded number of variables. As one considers more and more general base classes, one needs to construct more and more complex obstructions to prove lower bounds. For example, as instances of the base class no longer have bounded diameter (of the incidence graph of the formula) or bounded clause length, neither have the obstructions one needs to consider. Such obstructions become increasingly hard to separate. Our separator obstructions can separate obstruction trees containing an unbounded number of variables from all potential future obstruction trees. We obtain backdoors of bounded depth by combining the strengths of separator obstructions and obstruction trees. We further introduce a *game-theoretic framework* to reason about backdoors of bounded depth. With this notion, we can compute winning strategies instead of explicitly constructing backdoors, greatly simplifying the presentation of our algorithms.

We provide the proofs of statements marked with \star in the full version.

2 Preliminaries

Satisfiability. A *literal* is a propositional variable x or a negated variable $\neg x$. A *clause* is a finite set of literals that does not contain a complementary pair x and $\neg x$ of literals. A propositional formula in conjunctive normal form, or *CNF formula* for short, is a set of clauses. We denote by \mathcal{CNF} the class of all CNF formulas. Let $F \in \mathcal{CNF}$ and $c \in F$. We denote by $\text{var}(c)$ the set of all variables occurring in c , i.e., $\text{var}(c) = \{x \mid x \in c \vee \neg x \in c\}$ and we set $\text{var}(F) = \bigcup_{c \in F} \text{var}(c)$. For a set of literals L , we denote by $\bar{L} = \{\neg l \mid l \in L\}$, the set of complementary literals of the literals in L . The *size* of a CNF formula F is $\|F\| = \sum_{c \in F} |c|$.

Let $\tau : X \rightarrow \{0, 1\}$ be an assignment of some set X of propositional variables. If $X = \{x\}$ and $\tau(x) = \epsilon$, we will sometimes also denote the assignment τ by $x = \epsilon$ for brevity. We denote by $\text{true}(\tau)$ ($\text{false}(\tau)$) the set of all literals satisfied (falsified) by τ , i.e., $\text{true}(\tau) = \{x \in X \mid \tau(x) = 1\} \cup \{\neg x \in \bar{X} \mid \tau(x) = 0\}$ ($\text{false}(\tau) = \overline{\text{true}(\tau)}$). We denote by $F[\tau]$ the formula obtained from F after removing all clauses that are satisfied by τ and from the remaining clauses removing all literals that are falsified by τ , i.e., $F[\tau] = \{c \setminus \text{false}(\tau) \mid c \in F \text{ and } c \cap \text{true}(\tau) = \emptyset\}$. We say that an assignment satisfies F if $F[\tau] = \emptyset$. We say that F is *satisfiable* if there is some assignment $\tau : \text{var}(F) \rightarrow \{0, 1\}$ that satisfies F , otherwise F is *unsatisfiable*. CNFSAT denotes the propositional satisfiability problem, which takes as instance a CNF formula, and asks whether the formula is satisfiable.

The *incidence graph* of a CNF formula F is the bipartite graph G_F whose vertices are the variables and clauses of F , and where a variable x and a clause c are adjacent if and only if $x \in \text{var}(c)$. We identify a subgraph G' of the incidence graph G_F with the formula F' consisting of all the clauses of F that are in G' , each restricted to the adjacent variables in G' . With slight abuse of notation, we define $\text{var}(F')$ to be the variables occurring in G' . Via incidence graphs, graph theoretic concepts directly translate to CNF formulas. For instance, we say that F is *connected* if G_F is connected, and F' is a *connected component* of F if F' is a maximal connected subset of F . $\text{Conn}(F)$ denotes the set of connected components of F . We will also consider the *primal graph* of a CNF formula F , which has as vertex set $\text{var}(F)$, and has pairs of variables $x, y \in \text{var}(F)$ adjacent if and only if $x, y \in \text{var}(c)$ for some $c \in F$.

Base classes. Let $\alpha \subseteq \{+, -\}$ with $\alpha \neq \emptyset$, let $F \in \mathcal{CNF}$ and $c \in F$. We say that a literal l is an α -*literal* if it is a positive literal and $+$ $\in \alpha$ or it is a negative literal and $- \in \alpha$. We say that a variable v α -*occurs* in a clause c , if v or $\neg v$ is an α -literal that is contained in c . We denote by $\text{var}_\alpha(c)$ the set of variables that α -occur in c . For $\alpha \subseteq \{+, -\}$ with $\alpha \neq \emptyset$ and $s \in \mathbb{N}$, let $\mathcal{C}_{\alpha, s}$ be the class of all CNF formulas F such that every clause of F contains at most s α -literals. For $\mathcal{C} \subseteq \mathcal{CNF}$, we say that a clause c is \mathcal{C} -*good* if $\{c\} \in \mathcal{C}$. Otherwise, c is \mathcal{C} -*bad*. Let τ be any (partial) assignment of the variables of F . We will frequently make use of the fact that $\mathcal{C}_{\alpha, s}$ is *closed under assignments*, i.e., if $F \in \mathcal{C}_{\alpha, s}$, then also $F[\tau] \in \mathcal{C}_{\alpha, s}$. Therefore, whenever a clause $c \in F$ is $\mathcal{C}_{\alpha, s}$ -good it will remain $\mathcal{C}_{\alpha, s}$ -good in $F[\tau]$ and conversely whenever a clause is $\mathcal{C}_{\alpha, s}$ -bad in $F[\tau]$ it is also $\mathcal{C}_{\alpha, s}$ -bad in F .

The classes $\mathcal{C}_{\alpha, s}$ capture (according to Schaefer's Dichotomy Theorem [25]) the largest syntactic classes of CNF formulas for which the satisfiability problem can be solved in polynomial time: The class $\mathcal{C}_{\{+\}, 1} = \text{HORN}$ of *Horn formulas*, the class of $\mathcal{C}_{\{-\}, 1} = \text{DHORN}$ of *dual Horn formulas*, and the class $\mathcal{C}_{\{+, -\}, 2} = \text{KROM}$ of *Krom* (or *2CNF*) *formulas*. Note also that the class NULL of formulas containing no variables considered by Mählmann et al. [17] is equal to $\mathcal{C}_{\{+, -\}, 0}$. We follow Williams et al. [26] and focus on classes that are closed under assignments and therefore we do not consider the classes of 0/1-valid and affine formulas.

Note that every class $\mathcal{C}_{\alpha,s}$ (and therefore also the classes of Krom, Horn, and dual Horn formulas) is trivially *linear-time recognizable*, i.e., membership in the class can be tested in linear-time. We say that a class \mathcal{C} of formulas is *tractable* or *linear-time tractable*, if CNFSAT restricted to formulas in \mathcal{C} can be solved in polynomial-time or linear-time, respectively. The classes HORN, DHORN, KROM are linear-time tractable [1, 5].

3 Backdoor Depth

A *binary decision tree* is a rooted binary tree T . Every inner node t of T is assigned a propositional variable, denoted by $\text{var}(t)$, and has exactly one left and one right child, which corresponds to setting the variable to 0 or 1, respectively. Moreover, every variable occurs at most once on any root-to-leaf path of T . We denote by $\text{var}(T)$ the set of all variables assigned to any node of T . Finally, we associate with each node t of T , the truth assignment τ_t that is defined on all the variables $\text{var}(P) \setminus \{\text{var}(t)\}$ occurring on the unique path P from the root of T to t such that $\tau_t(v) = 0$ ($\tau_t(v) = 1$) if $v \in \text{var}(P) \setminus \{\text{var}(t)\}$ and P contains the left child (right child) of the node t' on P with $\text{var}(t') = v$. Let \mathcal{C} be a base class, F be a CNF formula, and T be a decision tree with $\text{var}(T) \subseteq \text{var}(F)$. Then T is a *\mathcal{C} -backdoor tree* of F if $F[\tau_t] \in \mathcal{C}$ for every leaf t of T [23].

Component backdoor trees generalize backdoor trees as considered by Samer and Szeider [23] by allowing an additional node type, *component nodes*, where the current instance is split into connected components. More precisely, let \mathcal{C} be a base class and F a CNF formula. A *component \mathcal{C} -backdoor tree* for F is a pair (T, φ) , where T is a rooted tree and φ is a mapping that assigns each node t a CNF formula $\varphi(t)$ such that the following conditions are satisfied:

1. For the root r of T , we have $\varphi(r) = F$.
2. For each leaf ℓ of T , we have $\varphi(\ell) \in \mathcal{C}$.
3. For each non-leaf t of T , there are two possibilities:
 - a. t has exactly two children t_0 and t_1 , where for some variable $x \in \text{var}(\varphi(t))$ we have $\varphi(t_i) = \varphi(t)[x = i]$; in this case we call t a *variable node*.
 - b. $\text{Conn}(\varphi(t)) = \{F_1, \dots, F_k\}$ for $k \geq 2$ and t has exactly k children t_1, \dots, t_k with $\varphi(t_i) = F_i$; in this case we call t a *component node*.

Thus, a backdoor tree is just a component backdoor tree without component nodes. The *depth* of a \mathcal{C} -backdoor is the largest number of variable nodes on any root-to-leaf path. The *\mathcal{C} -backdoor depth* $\text{depth}_{\mathcal{C}}(F)$ of a formula F into a base class \mathcal{C} is the smallest depth over all component \mathcal{C} -backdoor trees of F . Alternatively, we can define \mathcal{C} -backdoor depth recursively as in equation (1) from the introduction. For a component backdoor tree (T, φ) let $\text{var}(T, \varphi)$ be the set of all variables x such that some variable node t of T branches on x . We observe that one can use component \mathcal{C} -backdoor trees to decide the satisfiability of a formula.

► **Lemma 1** (\star). *Let $\mathcal{C} \subseteq \text{CNF}$ be tractable, let $F \in \text{CNF}$, and let (T, φ) be a component \mathcal{C} -backdoor tree of F of depth d . Then, we can decide the satisfiability of F in time $(2^d \|F\|)^{\mathcal{O}(1)}$. Moreover, if \mathcal{C} is linear-time tractable, then the same can be done in time $\mathcal{O}(2^d \|F\|)$.*

Let $\mathcal{C} \subseteq \text{CNF}$ and $F \in \text{CNF}$. A (strong) *\mathcal{C} -backdoor* of F is a set $B \subseteq \text{var}(F)$ such that $F[\tau] \in \mathcal{C}$ for each $\tau : B \rightarrow \{0, 1\}$. Assume \mathcal{C} is closed under partial assignments (which is the case for many natural base classes and the classes $\mathcal{C}_{\alpha,s}$) and (T, φ) a component \mathcal{C} -backdoor tree of F . Then $\text{var}(T, \varphi)$ is a \mathcal{C} -backdoor of F .

4 Technical Overview

We present all our algorithms in this work within a game-theoretic framework. This framework builds upon the following equivalent formulation of backdoor depth using splitter games. Similar games can be used to describe treedepth and other graph classes [15].

► **Definition 2.** Let $\mathcal{C} \subseteq \text{CNF}$ and $F \in \text{CNF}$. We denote by $\text{GAME}(F, \mathcal{C})$ the so-called \mathcal{C} -backdoor depth game on F . The game is played between two players, the connector and the splitter. The positions of the game are CNF formulas. At first, the connector chooses a connected component of F to be the starting position of the game. The game is over once a position in the base class \mathcal{C} is reached. We call these positions the winning positions (of the splitter). In each round the game progresses from a current position J to a next position as follows:

- the splitter chooses a variable $v \in \text{var}(J)$.
- The connector chooses an assignment $\tau: \{v\} \rightarrow \{0, 1\}$ and a connected component J' of $J[\tau]$. The next position is J' .

In the (unusual) case that a position J contains no variables anymore but J is still not in \mathcal{C} , the splitter loses. For a position J , we denote by τ_J the assignment of all variables assigned up to position J .

The following observation follows easily from the definition of the game and the fact that the (strategy) tree obtained by playing all possible plays of the connector against a given d -round winning strategy for the splitter forms a component backdoor tree of depth d , and vice versa. In particular, the splitter choosing a variable v at position J corresponds to a variable node and the subsequent choice of the connector for an assignment τ of v and a component of $J[\tau]$ corresponds to a component node (and a subsequent variable or leaf node) in a component backdoor tree.

► **Observation 3.** The splitter has a strategy for the game $\text{GAME}(F, \mathcal{C})$ to reach within at most d rounds a winning position if and only if F has \mathcal{C} -backdoor depth at most d .

Using backdoor depth games, we no longer have to explicitly construct a backdoor. Instead, we present so called *splitter-algorithms* that play the backdoor depth game from the perspective of the splitter. The algorithms will have some auxiliary internal state that they modify with each move. Formally, a splitter-algorithm for the \mathcal{C} -backdoor depth game to a base class \mathcal{C} is a procedure that

- gets as input a (non-winning) position J of the game, together with an internal state
- and returns a valid move for the splitter at position J , together with an updated internal state.

We will usually use the internal state to hold an obstruction that the splitter will periodically increase in size. Assume we have a game $\text{GAME}(F, \mathcal{C})$ and some additional input S . For a given strategy of the connector, the splitter-algorithm plays the game as one would expect: In the beginning, the internal state is initialized with S (if no additional input is given, the state is initialized empty). Whenever the splitter should make its next move, the splitter-algorithm is queried using the current position and internal state, and afterwards the internal state is updated accordingly.

► **Definition 4.** We say a *splitter-algorithm* implements a strategy to reach for a game $\text{GAME}(F, \mathcal{C})$ and input S within at most d rounds a position and internal state with some property if and only if initializing the internal state with S and then playing $\text{GAME}(F, \mathcal{C})$ according to the splitter-algorithm leads – no matter what strategy the connector is using – after at most d rounds to a position and internal state with said property.

The following observation converts splitter-algorithms into algorithms for bounded depth backdoors. It builds component backdoor trees by trying all moves of the connector.

► **Lemma 5** (\star). *Let $\mathcal{C} \subseteq \text{CNF}$ and $f_{\mathcal{C}}: \mathbb{N} \rightarrow \mathbb{N}$. Assume there exists a splitter-algorithm that implements a strategy to reach in each play in the game $\text{GAME}(F, \mathcal{C})$ and non-negative integer d within at most $f_{\mathcal{C}}(d)$ rounds either:*

- i) *a winning position, or*
- ii) *(an internal state representing) a proof that the \mathcal{C} -backdoor depth of F is at least d .*

Further assume this splitter-algorithm always takes at most $\mathcal{O}(\|F\|)$ time to compute its next move. Then there is an algorithm that, given F and d , in time at most $3^{f_{\mathcal{C}}(d)} \mathcal{O}(\|F\|)$ either:

- i) *returns a component \mathcal{C} -backdoor tree of depth at most $f_{\mathcal{C}}(d)$, or*
- ii) *concludes that the \mathcal{C} -backdoor depth of F is at least d .*

For the sake of readability, we may present splitter-algorithms as continuously running algorithms that periodically output moves (via some output channel) and always immediately as a reply get the next move of the connector (via some input channel). Such an algorithm can easily be converted into a procedure that gets as input a position and internal state and outputs a move and a modified internal state: The internal state encodes the whole state of the computation, (e.g., the current state of a Turing machine together with the contents of the tape and the position of the head). Whenever the procedure is called, it “unfreezes” this state, performs the computation until it reaches its next move and then “freezes” and returns its state together with the move.

Our main result is an approximation algorithm (Theorem 15) that either concludes that there is no backdoor of depth d , or computes a component backdoor tree of depth at most $2^{2^{\mathcal{O}(d)}}$. By Lemma 5, this is equivalent to a splitter-algorithm that plays for $2^{2^{\mathcal{O}(d)}}$ rounds to either reach a winning position or a proof that the backdoor depth is larger than d .

Following the approach of Mählmann et al. [17], our proofs of high backdoor depth come in the form of so-called *obstruction trees*. These are trees in the incidence graph of a CNF formula. Their node set therefore consists of both variables and clauses. Obstruction trees of depth d describe parts of an instance for which the splitter needs more than d rounds to win the backdoor depth game. For depth zero, we simply take a single (bad) clause that is not allowed by the base class. Roughly speaking, an obstruction tree of depth $d > 0$ is built from two “separated” obstruction trees T_1, T_2 of depth $d - 1$ that are connected by a path. Since the two obstruction trees are separated but in the same component, we know that for any choice of the splitter (i.e., choice of a variable v), there is a response of the connector (i.e., an assignment of v and a component) in which either T_1 or T_2 is whole. Then the splitter needs by induction still more than $d - 1$ additional rounds to win the game.

► **Definition 6.** *Let $F \in \text{CNF}$ and $\mathcal{C} = \mathcal{C}_{\alpha, s}$ with $\alpha \subseteq \{+, -\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. We inductively define \mathcal{C} -obstruction trees T for F of increasing depth.*

- *Let c be a \mathcal{C} -bad clause of F . The set $T = \{c\}$ is a \mathcal{C} -obstruction tree in F of depth 0.*
- *Let T_1 be a \mathcal{C} -obstruction tree of depth i in F . Let β be a partial assignment of the variables in F . Let T_2 be an obstruction tree of depth i in $F[\beta]$ such that no variable $v \in \text{var}(F[\beta])$ occurs both in a clause of T_1 and T_2 . Let further P be (a CNF formula representing) a path that connects T_1 and T_2 in F . Then $T = T_1 \cup T_2 \cup \text{var}(P) \cup P$ is a \mathcal{C} -obstruction tree in F of depth $i + 1$.*

► **Lemma 7** (\star). *Let $F \in \text{CNF}$ and $\mathcal{C} = \mathcal{C}_{\alpha, s}$ with $\alpha \subseteq \{+, -\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. If there is a \mathcal{C} -obstruction tree of depth d in F , then the \mathcal{C} -backdoor depth of F is larger than d .*

Our splitter-algorithm will construct obstruction trees of increasing depth by a recursive procedure (Lemma 14) that we outline now. We say a splitter-algorithm satisfies *property i* if it reaches in each game $\text{GAME}(F, \mathcal{C})$ within $g_{\mathcal{C}}(i, d)$ rounds (for some function $g_{\mathcal{C}}(i, d)$) either

- 1) a winning position, or
- 2) a position J and a \mathcal{C} -obstruction tree T of depth i in F such that no variable in $\text{var}(J)$ occurs in a clause of T , or
- 3) a proof that the \mathcal{C} -backdoor depth of F is at least d .

If we have a splitter algorithm satisfying property $d+1$ then our main result, the approximation algorithm for backdoor depth, directly follows from Lemma 7 and Lemma 5. Assume we have a strategy satisfying property $i-1$, let us describe how to use it to satisfy property i . If at any point we reach a winning position, or a proof that the \mathcal{C} -backdoor depth of F is at least d , we are done. Let us assume this does not happen, so we can focus on the much more interesting second case.

We use property $i-1$ to construct a first tree T_1 of depth $i-1$, and reach a position J_1 . We use it again, starting at position J_1 to construct a second tree T_2 of depth $i-1$ that is completely contained in position J_1 . Since in the beginning the connector selected a connected component, T_1 and T_2 are in the same component of F and we can find a path P connecting them. Let β be the assignment that assigns all the variables the splitter chose until reaching position J_1 . Then T_2 is an obstruction tree not only in J_1 but also in $F[\beta]$. In order to join both trees together into an obstruction of depth i , we have to show, according to Definition 6 that no variable $v \in \text{var}(F[\beta])$ occurs both in a clause of T_1 and T_2 . Since no variable in $\text{var}(J_1)$ occurs in a clause of T_1 (property $i-1$), and T_2 was built only from J_1 , this is the case. The trees T_1 and T_2 are “separated” and can be safely joined into a new obstruction tree T of depth i (see also Figure 3 on page 15 and the proof of Lemma 14 for details).

The last thing we need to ensure is that we reach a position J such that no variable in $\text{var}(J)$ occurs in a clause of T . This then guarantees that T is “separated” from all future obstruction trees that we may want to join it with to satisfy property $i+1$, $i+2$ and so forth. This is the major difficulty and main technical contribution of this paper.

It is important to note here, that the exact notion of “separation” between obstruction trees plays a crucial role for our approach and is one of the main differences to Mählmann et al. [17]. Mählmann et al. solve the separation problem in a “brute-force” manner: If we translate their approach to the language of splitter-algorithms, then the splitter simply selects all variables that occur in a clause of T . For their base class – the class NULL of formulas without variables – there are at most $2^{\mathcal{O}(d)}$ variables that occur in an obstruction tree of depth d . Thus, in only $2^{\mathcal{O}(d)}$ rounds, the splitter can select all of them, fulfilling the separation property. This completes the proof for the base class NULL.

However, already for backdoor depth to KROM, this approach cannot work since instances in the base class have obstruction trees with arbitrarily many clauses. Moreover, the situation becomes even more difficult for backdoors to HORN, since additionally clauses are allowed to contain arbitrary many literals. Mählmann et al. acknowledge this as a central problem and ask for an alternative approach to the separation problem that works for more general base classes.

5 Separator Obstructions

The main technical contribution of this work is a separation technique that works for the base classes $\mathcal{C} = \mathcal{C}_{\alpha, s}$. The separation technique is based on a novel form of obstruction, which we call *separator obstruction*. Obstruction trees are made up of paths, therefore, it is

sufficient to separate each new path P that is added to an obstruction. Note that P can be arbitrarily long and every clause on P can have arbitrary many variables and therefore the splitter cannot simply select all variables in (clauses of) P . Instead, given such a path P that we want to separate, we will use separator obstructions to develop a splitter-algorithm (Lemma 12) that reaches in each game $\text{GAME}(F, \mathcal{C})$ within a bounded number of rounds either

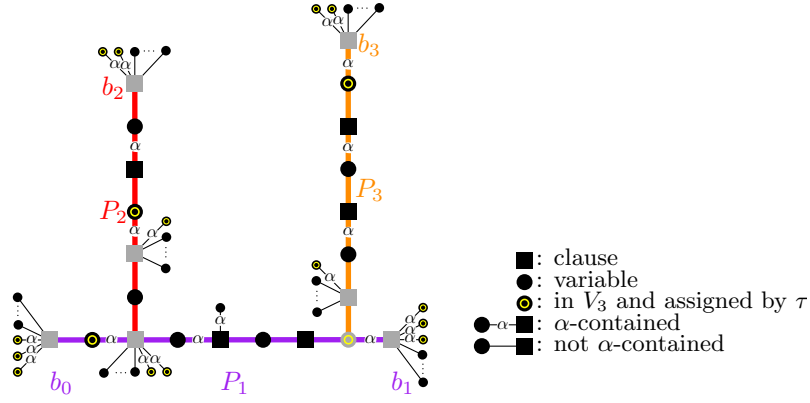
- 1) a winning position, or
- 2) a position J such that no variable in $\text{var}(J)$ occurs in a clause of P , or
- 3) a proof that the \mathcal{C} -backdoor depth of F is at least d .

Informally, a separator obstruction is a sequence $\langle P_1, \dots, P_\ell \rangle$ of paths that form a tree T_ℓ together with an assignment τ of certain *important* variables occurring in T_ℓ . The variables of τ correspond to the variables chosen by the splitter-algorithm and the assignment τ corresponds to the assignment chosen by the connector. Each path P_i adds at least one \mathcal{C} -bad clause b_i to the separator obstruction, which is an important prerequisite to increase the backdoor depth by growing the obstruction. Moreover, by choosing the important variables and the paths carefully, we ensure that for every *outside* variable, i.e., any variable that is not an important variable assigned by τ , there is an assignment and a component (which can be chosen by the connector) that leaves a large enough part of the separator obstruction intact. Thus, if a separator obstruction is sufficiently large, the connector can play such that even after d rounds a non-empty part of the separator obstruction is still intact. This means a large separator obstruction is a proof that the backdoor depth is larger than d .

To illustrate the growth of a separator obstruction (and motivate its definition) suppose that our splitter-algorithm is at position J of the game $\text{GAME}(F, \mathcal{C})$ and has already built a separator obstruction $X = \langle \langle P_1, \dots, P_i \rangle, \tau \rangle$ (with corresponding tree T_i) containing \mathcal{C} -bad clauses b_1, \dots, b_i ; note that τ is compatible with τ_J (i.e., τ and τ_J agree on the common assigned variables). If J is already a winning position, then property i is satisfied. Therefore, J has to contain a \mathcal{C} -bad clause. If no \mathcal{C} -bad clause has a path to T_i in J , then J satisfies 2) of property i and we are also done. Otherwise, let b_{i+1} be a \mathcal{C} -bad clause in J that is closest to T_i and let P_{i+1} be a shortest path from b_{i+1} to T_i in J . Then, we extend our separator obstruction X by attaching the path P_{i+1} to T_i (and obtain the tree T_{i+1}). Our next order of business is to choose a bounded number of important variables occurring on P_{i+1} that we will add to X . Those variables need to be chosen such that no outside variable can destroy too much of the separator obstruction. Apart from destroying the paths of the separator obstruction, we also need to avoid that assigning any outside variable makes too many of the \mathcal{C} -bad clauses b_1, \dots, b_{i+1} \mathcal{C} -good. Therefore, a natural choice would be to add all variables of b_{i+1} to X , i.e., to make those variables important. Unfortunately, this is not possible since b_{i+1} can contain arbitrarily many literals. Instead, we will only add the variables of b_{i+1} to X that α -occur in b_{i+1} . By the following lemma, the number of those variables is bounded.

► **Lemma 8** (\star). *Let $F \in \mathcal{CNF}$ and $\mathcal{C} = \mathcal{C}_{\alpha, s}$ with $\alpha \subseteq \{+, -\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. If F has \mathcal{C} -backdoor depth at most some integer d , then every clause of F contains at most $d + s$ α -literals.*

While this still allows for outside variables to occur in many of the \mathcal{C} -bad clauses b_1, \dots, b_{i+1} , it already ensures that no outside variable can α -occur in any of these clauses. This helps us, since when $|\alpha| = 1$ (i.e., the only case where α -occurs means something different than just occurs), it provides us with an assignment of any such outside variable that the connector can play without making the \mathcal{C} -bad clauses in which it occurs \mathcal{C} -good. For instance, if $\alpha = \{+\}$, then any outside variable v can only occur negatively in a \mathcal{C} -bad clause and moreover setting v to 0 ensures that the \mathcal{C} -bad clauses remain \mathcal{C} -bad.



■ **Figure 1** A separator obstruction containing three paths P_1 , P_2 , and P_3 . The figure shows vertices and edges of the incidence graph. Only the colorful edges are part of separator obstruction's tree. Gray variables and clauses are mentioned under the names b_i , e , and c in Definition 9.

Next, we need to ensure that any outside variable cannot destroy too many paths. By choosing a *shortest* path P_{i+1} , we have already ensured that no variable occurs on more than two clauses of P_{i+1} (such a variable would be a shortcut, meaning P_{i+1} was not a shortest path). Moreover, because P_{i+1} is a shortest path from b_{i+1} to T_i , every variable that occurs on T_i and on P_{i+1} must occur in the clause c in P_{i+1} that is closest to T_i but not in T_i itself. Similarly, to how we dealt with the \mathcal{C} -bad clauses, we will now add all variables that α -occur in c to X . This ensures that no outside variable can α -occur in both T_i and P_{i+1} , which (by induction over i) implies that every outside variable α -occurs in at most two clauses (either from T_i or from P_{i+1}) and therefore provides us with an assignment for the outside variables that removes at most two clauses from X . However, since removing any single clause can be arbitrarily bad if the clause has a high degree in the separator obstruction, we further need to ensure that all clauses of the separator obstruction in which outside variables α -occur have small degree. We achieve this by adding the variables α -occurring in any clause as soon as its degree (in the separator obstruction) becomes larger than two, which happens whenever the endpoint of P_{i+1} in T_i is a clause. Finally, if the endpoint of P_{i+1} in T_i is a variable, we also add this variable to the separator obstruction to ensure that no variable has degree larger than three in T_{i+1} . This leads us to the following definition of separator obstructions (see also Figure 1 for an illustration).

► **Definition 9.** Let $F \in \mathcal{CNF}$ and $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+, -\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. A \mathcal{C} -separator obstruction for F is a tuple $X = \langle (P_1, \dots, P_\ell), \tau \rangle$ (where P_1, \dots, P_ℓ are paths in F and τ is an assignment of variables of F) satisfying the following recursive definition.

- P_1 is a shortest path between two \mathcal{C} -bad clauses b_0 and b_1 in F . Let $B_1 = \{b_0, b_1\}$, let V_1 be the set of all variables that α -occur in any clause in B_1 , let $\tau_1 : V_1 \rightarrow \{0, 1\}$ be any assignment of the variables in V_1 , and let $T_1 = P_1$.
- For every i with $1 < i \leq \ell$, let b_i be a \mathcal{C} -bad clause in $F[\tau_{i-1}]$ of minimal distance to T_{i-1} in $F[\tau_{i-1}]$. Then, P_i is a shortest path (of possibly length zero) in $F[\tau_{i-1}]$ from T_{i-1} to b_i and $T_i = T_{i-1} \cup P_i$. Moreover, let e be the variable or clause that is both in T_{i-1} and P_i . We define B_i and V_i by initially setting $B_i = B_{i-1} \cup \{b_i\}$ and $V_i = V_{i-1} \cup \text{var}_\alpha(b_i)$ and distinguishing two cases:
 - If e is a variable, then let c be the clause on P_i incident with e (note that it is possible that $c = b_i$). Then, we add c to B_i and we add $\{e\} \cup \text{var}_\alpha(c)$ to V_i .

- If e is a clause, then either $e = b_i$ or $e \neq b_i$ and there is a clause c that is closest to e on P_i (it may be that $c = b_i$). In the former case we leave B_i and V_i unchanged and in the latter case, we add e and c to B_i and we add $\text{var}_\alpha(e) \cup \text{var}_\alpha(c)$ to V_i .
- $\tau_i : V_i \rightarrow \{0, 1\}$ is any assignment of the variables in V_i that is compatible with τ_{i-1} .

We set $\tau = \tau_\ell$. The size of X is the number of paths in $T = T_\ell$, i.e., $\ell + 1$.

The assignment τ is a central part of the definition, guiding the connector in Lemma 11 and thereby establishing a lower bound on the backdoor depth. We start by observing some simple but important properties of separator obstructions.

► **Lemma 10** (\star). *Let $F \in \text{CNF}$, $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+, -\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$, and let $X = \langle \langle P_1, \dots, P_\ell \rangle, \tau \rangle$ be a \mathcal{C} -separator obstruction in F , then for every $i \in [\ell]$:*

- (C1) T_i is a tree.
- (C2) Every variable $v \notin V_i$ occurs in at most two clauses of P_j for every j with $1 \leq j \leq i$ and moreover those clauses are consecutive in P_j .
- (C3) Every variable $v \notin V_i$ α -occurs in at most two clauses of T_i and moreover those clauses are consecutively contained in one path of T_i .
- (C4) Every variable $v \in V_i \setminus V_{i-1}$ α -occurs in most four clauses of T_i .
- (C5) If a variable $v \notin V_i$ α -occurs in a clause c of T_i , then c has degree at most two in T_i .
- (C6) Every variable of F has degree at most three in T .
- (C7) If every clause of F contains at most x α -literals, then $|V_i \setminus V_{i-1}| \leq 2s + x + 1$.

We now show the main result of this subsection, namely, that also separator obstructions can be used to obtain a lower bound on the backdoor depth of CNF formulas.

► **Lemma 11.** *Let $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+, -\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$ and $F \in \text{CNF}$. If F has a \mathcal{C} -separator obstruction of size at least $\ell = (8^d(14^2 + 2d))^{2^d}$, then F has \mathcal{C} -backdoor depth at least d .*

Proof. Let $X = \langle \langle P_1, \dots, P_\ell \rangle, \tau \rangle$ be a \mathcal{C} -separator obstruction for F of size at least ℓ with V_i, B_i, T_i, T as in Definition 9. Let J be a position in the game $\text{GAME}(F, \mathcal{C})$. We say that a subtree T' of $T = T_\ell$ is *contained* in J if every variable and clause of T' occurs in J . Let T' be a subtree of T that is contained in J . Let P_j be a path of X . We say that P_j is *active* in T' if either $V(P_j) = \{b_j\}$ and T' contains b_j or T' contains a vertex in $V(P_j) \setminus V(T_{j-1})$. Moreover, we say that P_j is *intact* in T' at position J if $V(P_j) \subseteq V(T')$ and b_j is a \mathcal{C} -bad clause in J . Otherwise, we say that P_j is *broken* in T' at position J .

We show by induction on the number of rounds that there is a strategy \mathbf{S} for the connector such that the following holds for every position J reached after i rounds in the game $\text{GAME}(F, \mathcal{C})$ against \mathbf{S} : At position J , there is a subtree T' of T contained in J that contains at least $\ell_i = (\ell^{1/2})^i / 8^i$ intact paths and at most $z_i = 2i$ broken paths of X . This then shows the statement of the lemma because $\ell_d = \ell^{1/2^d} / 8^d = 14^2 + 2d \geq 1$ and therefore any position J reached after d rounds in the game $\text{GAME}(F, \mathcal{C})$ contains at least one clause that is \mathcal{C} -bad in J .

The claim clearly holds for $i = 0$ since $\ell_0 = \ell$ and $z_0 = 0$ and the connector can choose the component of F containing T . Assume now that $i > 0$ and let J be the position reached after $i - 1$ rounds. By the induction hypothesis, at position J there is a subtree T' of T contained in J containing at least $\ell_{i-1} = \ell^{(1/2)^{i-1}} / 8^{i-1}$ intact paths and at most $z_{i-1} = 2(i - 1)$ broken paths of X . Suppose that the splitter chooses variable v as its next move. Moreover, let o be the smallest integer such that $v \in V_o$; if $v \notin V_\ell$ we set $o = \ell + 1$. Note that $v \notin V_j$ for every

$i < o$. Let I be the set of all paths P_j of X that are intact in T' at position J and let $I_{<o}$ ($I_{>o}$) be the subset of I containing only the paths P_j with $j < o$ ($j > o$). Finally, let $T'_{<o}$ be the subtree of T' restricted to the paths P_j of X with $j < o$. Note that at position J , $T'_{<o}$ is connected and the paths in $I_{<o}$ are intact also in $T'_{<o}$. Then, the connector chooses the assignment $\beta : \{v\} \rightarrow \{0, 1\}$ such that:

$$\beta(v) = \begin{cases} \tau(v) & |I_{<o}| < \sqrt{\ell_{i-1}}, \\ 1 & |I_{<o}| \geq \sqrt{\ell_{i-1}} \text{ and } + \in \alpha, \\ 0 & \text{otherwise.} \end{cases}$$

As we will show below, β is defined in such a manner that the position $J' = J[\beta]$ reached after the next round of the game $\text{GAME}(F, \mathcal{C})$ contains a subtree T'' of T' containing at least $\ell_i = \sqrt{\ell_{i-1}}/8$ paths that are intact in J' and at most $z_i = z_{i-1} + 2$ broken paths, which completes the proof since the connector can now choose the component of J' containing T'' to fulfill the induction invariant. We distinguish the following cases; refer also to Figure 2 for an illustration of the two cases.

Case 1: $|I_{<o}| \geq \sqrt{\ell_{i-1}}$. We will show that T'' can be obtained as a subtree of $T'_{<o}$.

Note first that all clauses b_j with $j < o$ that are \mathcal{C} -bad in J are also \mathcal{C} -bad in J' . This is because $v \notin V_j$ (because $j < o$ and $v \notin V_{o-1}$) and therefore v cannot α -occur in b_j , which implies that b_j remains \mathcal{C} -bad and not satisfied after setting v to $\beta(v)$.

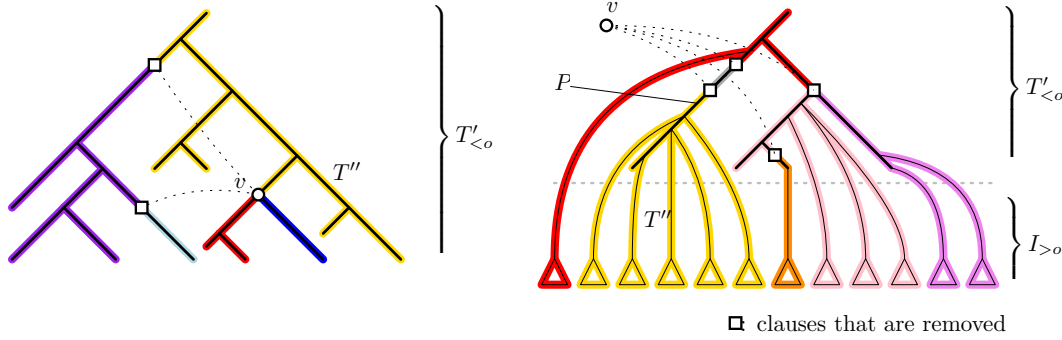
The tree $T'_{<o}$ in J may decompose into multiple components in J' . We will argue that one of these components contains many intact paths and only at most two more broken paths than $T'_{<o}$. Since the \mathcal{C} -bad clauses of an intact path remain \mathcal{C} -bad in J' , the only way in which an intact path can become broken is if parts of the path get removed, i.e., either v or clauses satisfied by setting v to $\beta(v)$.

If $\beta(v) = 1$ then $+ \in \alpha$. If $\beta(v) = 0$ then $+ \notin \alpha$, and since $\alpha \neq \emptyset$, then $- \in \alpha$. Thus, in $J' = J[\beta]$, the only elements that are removed are the variable v as well as clauses in which v α -occurs. By Lemma 10 (C3), v α -occurs in at most two clauses of $T'_{<o}$ and because of (C5) those clauses have degree at most two in $T'_{<o}$. Therefore, setting v to $\beta(v)$ removes at most two clauses from $T'_{<o}$, each of which having degree at most two. Moreover, according to Lemma 10 (C6), v itself has degree at most three in $T'_{<o}$. This implies that setting v to $\beta(v)$ splits $T'_{<o}$ into at most $2 \cdot 2 + 3 = 7$ components.

Moreover, because of Lemma 10 (C3), the at most two clauses of $T'_{<o}$ in which v α -occurs are located on the same path P_j . Therefore, at most two paths that are complete in $T'_{<o}$, i.e., the path P_j and the at most one path containing v , can become broken. Therefore, there is a component of J' that contains a subtree of $T'_{<o}$ that contains at least $|I_{<o}|/7 - 2 \geq \sqrt{\ell_{i-1}}/7 - 2$ intact paths and at most $z_{i-1} + 2 \leq 2i = z_i$ broken paths of X . Note that $\sqrt{\ell_{i-1}} \geq \ell_d \geq 14^2 + 2d \geq 14^2$ and therefore $\sqrt{\ell_{i-1}}/7 - 2 \geq \sqrt{\ell_{i-1}}/8 = \ell_i$.

Case 2: $|I_{<o}| < \sqrt{\ell_{i-1}}$. This means $\beta(v) = \tau(v)$. In this case, we will build the subtree T'' by picking only one path from $T'_{<o}$ and the remaining paths from P_{o+1}, \dots, P_ℓ . Let A be the set of all paths of X that are active in T' and let $A_{>o}$ ($A_{<o}$) be the subset of A containing only the paths P_j with $j > o$ ($j < o$). We say that a path P_a of X is *attached* to a path P_b of X if $a > b$, $V(P_a) \cap V(P_b) \neq \emptyset$ and there is no $b' < b$ with $V(P_a) \cap V(P_{b'}) \neq \emptyset$. We say that a path P_a in $A_{>o}$ is *weakly attached* to a path P_b in $A_{<o}$ if either:

- P_a is attached to P_b or
- P_a is attached to a path P_c in $A_{>o}$ that is in turn weakly attached to P_b .



■ **Figure 2** Left: Case 1. The set $I_{<o}$ is large. Assigning v to $\beta(v)$ decomposes the tree $T'_{<o}$ into at most seven components. The largest component T'' is still large. Right: Case 2. The set $I_{<o}$ is small. There is a path P to which many paths are weakly attached, forming a tree T_P . Assigning v to $\beta(v)$ splits T_P in at most three parts. The largest component T'' of T_P is still large.

Note that because T' is a tree, every path in $A_{>o}$ is weakly attached to exactly one path in $A_{<o}$. Moreover, for the same reason any path in $A_{<o}$ together with all paths in $A_{>o}$ that are weakly attached to it forms a subtree of T' .

Therefore, there is a path P in $A_{<o}$ such that at least $|I_{>o}|/|A_{<o}|$ paths in $I_{>o}$ are weakly attached to P . Moreover, the union T_P of P and all paths in $A_{>o}$ that are weakly attached to P is a subtree of T' . Note that T_P has at least $|I_{>o}|/|A_{<o}|$ paths that are intact in T_P and at most z_{i-1} paths that are broken in T_P at position J . Since $\sqrt{\ell_{i-1}} \geq \ell_d = 14^2 + 2d \geq 2d$ and $z_{i-1} \leq z_d = 2d$, it holds that $|I_{<o}| + z_{i-1} \leq 2\sqrt{\ell_{i-1}}$ (because also $|I_{<o}| \leq \sqrt{\ell_{i-1}}$). Therefore,

$$\begin{aligned}
 |I_{>o}|/|A_{<o}| &\geq (\ell_{i-1} - |I_{<o}|)/(|I_{<o}| + z_{i-1}) \\
 &\geq (\ell_{i-1} - |I_{<o}|)/(2|I_{<o}|) \\
 &\geq (\ell_{i-1})/(2\sqrt{\ell_{i-1}}) - 1/2 \\
 &\geq \sqrt{\ell_{i-1}}/2 - 1/2 \\
 &= 8\ell_i/2 - 1 \geq 3\ell_i.
 \end{aligned}$$

Because $\beta(v) = \tau(v)$, all paths P_j with $j > o$ that are active in T_P are still contained in J' and moreover if P_j is intact in J , then it is still intact in J' . Moreover, because of Lemma 10 (C2), v occurs in at most two clauses of P and because $\beta(v) = \tau(v)$ all paths P_{o+1}, \dots, P_ℓ that are attached to P are still attached to P after setting v to $\beta(v)$. It follows that setting v to $\beta(v)$ removes at most two clauses and at most one variable (i.e., the variable v) from P and also from T_P . Therefore, $J' = J[\beta]$ contains a component that contains a subtree T'' of T_P with at least $3\ell_i/3 = \ell_i$ paths that are intact in T'' and at most $z_{i-1} + 1 \leq z_i$ paths that are broken in T'' . ◀

6 Winning Strategies and Algorithms

We are ready to present our algorithmic results. Earlier, we discussed that separator obstructions are used to separate existing obstruction trees from future obstruction trees. As all obstruction trees are built only from shortest paths, it is sufficient to derive a splitter-algorithm that takes a shortest path P and separates it from all future obstructions. By reaching a position J such that no variable in $\text{var}(J)$ occurs in a clause of P , we are guaranteed that all future obstructions are separated from P , as future obstructions will only contain clauses and variables from J .

► **Lemma 12** (\star). Let $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+, -\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. There exists a splitter-algorithm that implements a strategy to reach for each game $\text{GAME}(F, \mathcal{C})$, non-negative integer d , and shortest path P between two \mathcal{C} -bad clauses in F within at most $(3s + d + 1)(8^d(14^2 + 2d))^{2^d}$ rounds either:

- 1) a winning position, or
- 2) a position J such that no variable in $\text{var}(J)$ occurs in a clause of P , or
- 3) a proof that the \mathcal{C} -backdoor depth of F is at least d .

This algorithm takes at most $\mathcal{O}(\|F\|)$ time per move.

Since selecting more variables can only help the splitter in archiving their goal, we immediately also get the following statement from Lemma 12.

► **Corollary 13**. Consider $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+, -\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$, a game $\text{GAME}(F, \mathcal{C})$ and a position J' in this game, a non-negative integer d and shortest path P between two \mathcal{C} -bad clauses in F . There exists a splitter-algorithm that implements a strategy that continues the game from position J' and reaches within at most $(3s + d + 1)(8^d(14^2 + 2d))^{2^d}$ rounds either:

- 1) a winning position, or
- 2) a position J such that no variable in $\text{var}(J)$ occurs in a clause of P , or
- 3) a proof that the \mathcal{C} -backdoor depth of F is at least d .

This algorithm takes at most $\mathcal{O}(\|F\|)$ time per move.

As described at the end of Section 4, we can now construct in the following lemma obstruction trees of growing size, using the previous corollary to separate them from potential future obstruction trees.

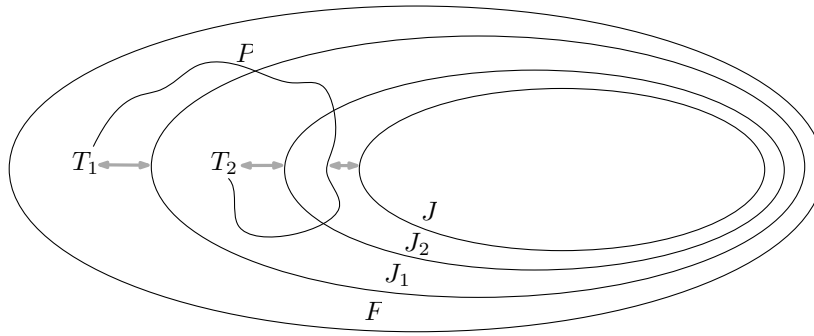
► **Lemma 14**. Let $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+, -\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. There is a splitter-algorithm that implements a strategy to reach for a game $\text{GAME}(F, \mathcal{C})$ and non-negative integers i, d with $1 \leq i \leq d$ within at most $(2^i - 1)(3s + d + 1)(8^d(14^2 + 2d))^{2^d}$ rounds either:

- 1) a winning position, or
- 2) a position J and a \mathcal{C} -obstruction tree T of depth i in F such that no variable in $\text{var}(J)$ occurs in a clause of T , or
- 3) a proof that the \mathcal{C} -backdoor depth of F is at least d .

This algorithm takes at most $\mathcal{O}(\|F\|)$ time per move.

Proof. We will prove this lemma by induction over i . Our splitter-algorithm will try construct an obstruction tree of depth i by first using the induction hypothesis to build two obstruction trees T_1 and T_2 of depth $i - 1$ and then joining them together. After the construction of the first tree T_1 , we reach a position J_1 and by our induction hypothesis no variable in $\text{var}(J_1)$ occurs in a clause of T_1 . This encapsulates the core idea behind our approach, as it means that T_1 is separated from all potential future obstruction trees T_2 that we build from position J_1 . Therefore, we can compute the next tree T_2 in J_1 and join T_1 and T_2 together in accordance with Definition 6 by a path P . At last, we use Corollary 13 to also separate this path from all future obstructions. If at any point of this process we reach a winning position or a proof that the \mathcal{C} -backdoor depth of F is at least d , we can stop. Let us now describe this approach in detail.

For convenience, let $x = (3s + d + 1)(8^d(14^2 + 2d))^{2^d}$. We start our induction with $i = 1$. If there is no \mathcal{C} -bad clause in F , then it is a winning position and we can stop. Assume there is exactly one \mathcal{C} -bad clause c in F . By Lemma 8, if c contains more than $d + s$ α -literals, we have a proof that the \mathcal{C} -backdoor depth of F is at least d and we archive case 3) of the lemma. On the other hand, if c contains at most $d + s$ α -literals, the splitter can obtain a



■ **Figure 3** Overview of the construction in Lemma 14. First, T_1 is chosen in F , yielding J_1 . Then, T_2 is chosen in J_1 , yielding J_2 . In the end the connecting path P is chosen yielding J . A gray doublesided arrow between a position \hat{J} and structure \hat{T} symbolizes that no variable $v \in \text{var}(\hat{J})$ occurs in a clause of \hat{T} .

winning position in $\text{GAME}(F, \mathcal{C})$ after at most $d + s \leq (2^i - 1)x$ rounds by choosing a new variable α -occurring in c at every round. Assume there is more than one \mathcal{C} -bad clause in F . Thus, we pick \mathcal{C} -bad clauses c_1 and c_2 and compute a shortest path P between c_1 and c_2 in F . By Definition 6, $T = \{c_1\} \cup \{c_2\} \cup \text{var}(P) \cup P$ is a \mathcal{C} -obstruction tree of depth 1 in F . We then continue the game using Corollary 13 (for the path P) to reach a position J' satisfying (1), (2), or (3) after at most $x \leq (2^i - 1)x$ rounds, with each round taking at most $\mathcal{O}(\|F\|)$ time.

We now assume the statement of this lemma to hold for $i - 1$ and we show it also holds for i . To this end, we start playing the game $\text{GAME}(F, \mathcal{C})$ according to the existing splitter-algorithm for $i - 1$. If we reach (within at most $(2^{i-1} - 1)x$ rounds) a winning position or a proof that the \mathcal{C} -backdoor depth of F is at least d then we are done. Assuming this is not the case, we reach a position J_1 and a \mathcal{C} -obstruction tree T_1 of depth $i - 1$ in F such that no variable $v \in \text{var}(J_1)$ occurs in a clause of T_1 .

We continue playing the game at position J_1 according to the existing splitter-algorithm for $\text{GAME}(J_1, \mathcal{C})$ and $i - 1$. The \mathcal{C} -backdoor depth of F is larger or equal to the \mathcal{C} -backdoor depth of J_1 . Thus again (after at most $(2^{i-1} - 1)x$ rounds) we either are done (because we reach a winning position or can conclude that the \mathcal{C} -backdoor depth of J_1 is at least d) or we reach a position J_2 and a \mathcal{C} -obstruction tree T_2 of depth $i - 1$ in J_1 such that no variable $v \in \text{var}(J_2)$ occurs in a clause of T_2 .

We pick two clauses $c_1 \in T_1$ and $c_2 \in T_2$ that are \mathcal{C} -bad in F and compute a shortest path P between c_1 and c_2 in F . We now argue that $T = T_1 \cup T_2 \cup \text{var}(P) \cup P$ is a \mathcal{C} -obstruction tree of depth i in F . Let $\beta = \tau_{J_1}$ be the assignment that assigns all the variables the splitter chose until reaching position J_1 to the value given by the connector. Note that J_1 is a connected component of $F[\beta]$.

Since all variables and clauses belonging to T_2 induce a connected subgraph of J_1 , T_2 is a \mathcal{C} -obstruction tree of depth $i - 1$ not only in J_1 , but also in $F[\beta]$. Let $v \in \text{var}(F[\beta])$. We show that v does not occur both in some clause of T_1 and of T_2 . To this end, assume v is contained in a clause of T_2 . Since all clauses of T_2 are in J_1 and J_1 is a connected component of $F[\beta]$, we further have $v \in \text{var}(J_1)$. On the other hand (as discussed earlier), no variable $v \in \text{var}(J_1)$ is contained in a clause of T_1 . By Definition 6, $T = T_1 \cup T_2 \cup \text{var}(P) \cup P$ is a \mathcal{C} -obstruction tree of depth i in F .

We use Corollary 13 to continue playing the game at position J_2 . Again, if we reach a winning position or a proof that the \mathcal{C} -backdoor depth of F is at least d we are done. So we focus on the third case that we reach (within at most x rounds) a position J such that

no variable $v \in \text{var}(J)$ is contained in a clause of P . We know already that no variable $v \in \text{var}(J_1)$ is contained in a clause of T_1 and no variable $v \in \text{var}(J_2)$ is contained in a clause of T_2 . Since $\text{var}(J) \subseteq \text{var}(J_2) \subseteq \text{var}(J_1)$, and $T = T_1 \cup T_2 \cup \text{var}(P) \cup P$, we can conclude that no variable $v \in \text{var}(J)$ is contained in a clause of T .

In total, we played for $(2^{i-1} - 1)x + (2^{i-1} - 1)x + x = (2^i - 1)x$ rounds. The splitter-algorithm in Corollary 13 takes at most $\mathcal{O}(\|F\|)$ time per move. The same holds for the splitter-algorithm for $i - 1$ that we use as a subroutine. Thus, the whole algorithm takes at most $\mathcal{O}(\|F\|)$ time per move. \blacktriangleleft

The main results now follow easily by combining Lemmas 1, 5, 7, and 14.

- **Theorem 15** (\star). *Let $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+, -\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. We can, for a given $F \in \text{CNF}$ and a non-negative integer d , in time at most $2^{2^{2^{\mathcal{O}(d)}}} \|F\|$ either*
- 1) *compute a component \mathcal{C} -backdoor tree of F of depth at most $2^{2^{\mathcal{O}(d)}}$, or*
 - 2) *conclude that the \mathcal{C} -backdoor depth of F is larger than d .*

► **Corollary 16.** *Let $\mathcal{C} \in \{\text{HORN}, \text{DHORN}, \text{KROM}\}$. The CNFSAT problem can be solved in linear time for any class of formulas of bounded \mathcal{C} -backdoor depth.*

7 Conclusion

We show that CNFSAT can be solved in linear-time for formulas of bounded \mathcal{C} -backdoor depth whenever \mathcal{C} is any of the well-known Schaefer classes. We achieve this by showing that \mathcal{C} -backdoor depth can be FPT-approximated for any class $\mathcal{C} = \mathcal{C}_{\alpha,s}$. This allows us to extend the results of Mählmann et al. [17] for the class of variable-free formulas to all Schaefer classes. Our results provide an important milestone towards generalizing and unifying the various tractability results based on variants of \mathcal{C} -backdoor size (see also future work below) to the only recently introduced and significantly more powerful \mathcal{C} -backdoor depth.

Let us finish with some natural and potentially significant extensions of backdoor depth that can benefit from our approach based on separator obstructions. Two of the probably most promising ones that have already been successfully employed as extensions of backdoor size are the so-called *scattered* and *heterogeneous* backdoor sets [11, 10].

Interestingly, while those two notions lead to orthogonal tractable classes in the context of backdoor size, they lead to the same tractable class for backdoor depth. Therefore, lifting these two extensions to backdoor depth, would result in a unified and significantly more general approach. While we are hopeful that our techniques can be adapted to this setting, one of the main remaining obstacles is that obstructions of depth 0 no longer are single (bad) clauses. For instance, consider the heterogeneous class $\mathcal{C} = \text{HORN} \cup \text{KROM}$. Here, a CNF formula may not be in \mathcal{C} due to a pair of clauses, one in $\text{HORN} \setminus \text{KROM}$ and another one in $\text{KROM} \setminus \text{HORN}$. Finally, an even more general but also more challenging tractable class to consider for backdoor depth is the class of Q-HORN formulas, which generalizes the heterogeneous class obtained as the union of all considered Schaefer classes.

References

- 1 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.
- 2 Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 340–351, 2003.

- 3 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016.
- 4 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Annual Symp. on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 1971.
- 5 William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 1(3):267–284, 1984.
- 6 Fedor V Fomin, Petr A Golovach, and Dimitrios M Thilikos. Parameterized complexity of elimination distance to first-order logic properties. *arXiv preprint arXiv:2104.02998*, 2021.
- 7 Vijay Ganesh and Moshe Y. Vardi. On the unreasonable effectiveness of SAT solvers. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 547–566. Cambridge University Press, 2020. doi:10.1017/9781108637435.032.
- 8 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Backdoor treewidth for SAT. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017 – 20th International Conference, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 20–37. Springer Verlag, 2017. doi:10.1007/978-3-319-66263-3_2.
- 9 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Combining treewidth and backdoors for CSP. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:17, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2017.36.
- 10 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. *ACM Transactions on Algorithms*, 13(2):29:1–29:32, 2017. Full version of a SODA’16 paper. doi:10.1145/3014587.
- 11 Serge Gaspers, Neeldhara Misra, Sebastian Ordyniak, Stefan Szeider, and Stanislav Zivny. Backdoors into heterogeneous classes of SAT and CSP. *J. of Computer and System Sciences*, 85:38–56, 2017. doi:10.1016/j.jcss.2016.10.007.
- 12 Serge Gaspers, Sebastian Ordyniak, M. S. Ramanujan, Saket Saurabh, and Stefan Szeider. Backdoors to q-Horn. *Algorithmica*, 74(1):540–557, 2016. doi:10.1007/s00453-014-9958-5.
- 13 Serge Gaspers and Stefan Szeider. Strong backdoors to bounded treewidth SAT. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26–29 October, 2013, Berkeley, CA, USA*, pages 489–498. IEEE Computer Society, 2013.
- 14 Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 89–134. Elsevier, 2008.
- 15 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.
- 16 Leonid Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.
- 17 Nikolas Mählmann, Sebastian Siebertz, and Alexandre Vigny. Recursive backdoors for SAT. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23–27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 73:1–73:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.73.
- 18 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.
- 19 Jaroslav Nešetřil and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European J. Combin.*, 27(6):1022–1041, 2006.
- 20 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.
- 21 Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *Proceedings of SAT 2004 (Seventh International Conference on Theory and Applications of Satisfiability Testing, 10–13 May, 2004, Vancouver, BC, Canada)*, pages 96–103, 2004.

- 22 Sebastian Ordyniak, Andre Schidler, and Stefan Szeider. Backdoor DNFs. In Zhi-Hua Zhou, editor, *Proceeding of IJCAI-2021, the 30th International Joint Conference on Artificial Intelligence*, pages 1403–1409, 2021. doi:10.24963/ijcai.2021/194.
- 23 Marko Samer and Stefan Szeider. Backdoor trees. In *AAAI 08, Twenty-Third Conference on Artificial Intelligence, Chicago, Illinois, July 13–17, 2008*, pages 363–368. AAAI Press, 2008.
- 24 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.
- 25 Thomas J. Schaefer. The complexity of satisfiability problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, Calif., 1978)*, pages 216–226. ACM, 1978.
- 26 Ryan Williams, Carla Gomes, and Bart Selman. Backdoors to typical case complexity. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI 2003*, pages 1173–1178. Morgan Kaufmann, 2003.

Finding a Cluster in Incomplete Data

Eduard Eiben  

Department of Computer Science, Royal Holloway, University of London, Egham, UK

Robert Ganian  

Algorithms and Complexity Group, TU Wien, Austria

Iyad Kanj  

School of Computing, DePaul University, Chicago, IL, USA

Sebastian Ordyniak  

University of Leeds, School of Computing, Leeds, UK

Stefan Szeider  

Algorithms and Complexity Group, TU Wien, Austria

Abstract

We study two variants of the fundamental problem of finding a cluster in incomplete data. In the problems under consideration, we are given a multiset of incomplete d -dimensional vectors over the binary domain and integers k and r , and the goal is to complete the missing vector entries so that the multiset of complete vectors either contains (i) a cluster of k vectors of radius at most r , or (ii) a cluster of k vectors of diameter at most r . We give tight characterizations of the parameterized complexity of the problems under consideration with respect to the parameters k , r , and a third parameter that captures the missing vector entries.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, incomplete data, clustering

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.47

Funding *Robert Ganian*: Project No. Y1329 of the Austrian Science Fund (FWF).

Sebastian Ordyniak: Project EP/V00252X/1 of the Engineering and Physical Sciences Research Council (EPSRC).

Stefan Szeider: Project No. P32441 of the Austrian Science Fund (FWF) and Project No. ICT19-065 of the Vienna Science and Technology Fund (WWTF).

1 Introduction

We consider two formulations of the fundamental problem of finding a sufficiently large cluster in incomplete data [2, 3, 18, 24]. In the setting under consideration, the input is a multiset M of d -dimensional Boolean vectors – regarded as the rows of a matrix, some of whose entries might be missing – and two parameters $k, r \in \mathbb{N}$. In the first problem under consideration, referred to as DIAM-CLUSTER-COMPLETION, the goal is to decide whether there is a completion of M that admits a multiset of k vectors (which we call a k -cluster) of diameter at most r ; that is, a k -cluster such that the Hamming distance between any two cluster-vectors is at most r . In the second problem, referred to as RAD-CLUSTER-COMPLETION, the goal is to decide whether there is a completion of M that admits a k -cluster of radius at most r ; that is, a k -cluster such that there is a *center* vector $\vec{s} \in \{0, 1\}^d$ with Hamming distance at most r to each cluster-vector.



© Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 47; pp. 47:1–47:14

Leibniz International Proceedings in Informatics



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

The cluster-diameter and cluster-radius are among the most widely used measures for intra-cluster similarity [5, 8, 12, 15, 16, 17, 18]. Our interest in studying these problems stems from the recent relevant research within the theory community [9, 14, 19, 20, 21], as well as the ubiquitous presence of incomplete data in relevant areas, such as recommender systems, machine learning, computer vision, and data science [1, 10, 11, 28].

We study the parameterized complexity of the above problems with respect to the two parameters k , r , and a third parameter that captures the occurrence of missing vector entries. Naturally, parameterizing by the number of missing entries alone is not desirable since one would expect that number to be rather large. In their recent related works on clustering problems, Koana, Froese, and Niedermeier [20, 21] restricted the occurrence of missing entries by using the maximum number of missing entries per row as the parameter. Another parameter for restricting the occurrence of missing entries is the minimum number of vectors plus coordinates needed to “cover” all missing entries, which was proposed and used by Eiben et al. [9] and Ganian et al. [14], who studied various data completion and clustering problems. In this paper, we propose and use a parameter that unifies and subsumes both previous parameterizations: the “*deletion distance to near-completion*”, denoted $\lambda(M)$, which is the minimum integer p such that at most p vectors can be removed from M so that every remaining vector contains at most p missing entries. Clearly, the parameter $\lambda(M)$ is computable in polynomial time and is not larger than any of (and hence subsumes) the two parameters considered by Koana et al. [20, 21], Eiben et al. [9], and Ganian et al. [14].

Results and Techniques. We perform an in-depth analysis of the two considered data completion problems w.r.t. the aforementioned parameterizations. We obtain results that provide a nearly complete complexity landscape of these problems. An overview of our results is provided in Table 1. As a byproduct, our results establish that both problems under consideration are fixed-parameter tractable parameterized by $k + r$ when the data is complete, which answers an open question in the literature [2, 3].

■ **Table 1** Overview of the results obtained in this paper.

	k	r	$k + r$	$k + \lambda$	$r + \lambda$	$k + r + \lambda$
DIAM-CLUSTER-C.	W[1]-h/XP	paraNP-c	W[1]-h/XP	W[1]-h/XP	FPT	FPT
RAD-CLUSTER-C.	W[1]-h/XP	paraNP-c	W[1]-h/XP	?/XP	?/XP	FPT

We summarize the new results obtained in this paper below.

1. We show that DIAM-CLUSTER-COMPLETION is fixed-parameter tractable (FPT) parameterized by $r + \lambda(M)$ (Theorem 9). The significance of the above result is in removing the dependency on the cluster size k in the running time of the algorithm, thus showing that finding a large cluster in incomplete data can be feasible when both the cluster diameter and the parameter $\lambda(M)$ are small. This result is the pinnacle of our technical contributions and relies on two ingredients: a fixed-parameter algorithm for the same problem parameterized by $k + r + \lambda(M)$ (Theorem 1), which is then used as a subroutine in the main algorithm, and a new technique that we dub *iterative sunflower harvesting*. Crucial to this new technique is a general structural lemma, allowing us to represent a family of sets in a succinct manner in terms of sunflower cores, which we believe to be interesting in its own right. We note that the use of sunflowers to obtain a succinct set representation (leading to a kernel) is not uncommon in such settings [20, 21, 22, 25]. What makes the sunflower harvesting technique novel is that it allows us to (1) show

that each solution can be covered by a small number of sunflowers, and to (2) iteratively “harvest” these sunflowers to obtain a solution in boundedly-many (in the parameter) branching steps.

2. We give an XP-algorithm for DIAM-CLUSTER-COMPLETION parameterized by k alone (Theorem 10). Together with Theorem 11 (showing the $W[1]$ -hardness of DIAM-CLUSTER w.r.t. k) and Theorem 12 (showing the $W[1]$ -hardness for DIAM-CLUSTER-COMPLETION w.r.t. k even for $r = 0$), this gives a complete complexity landscape for DIAM-CLUSTER-COMPLETION parameterized by any combination of the parameters k , r , and $\lambda(M)$.
3. We show that RAD-CLUSTER-COMPLETION is FPT parameterized by $k + r + \lambda(M)$ (Theorem 15); this result answers open questions in the literature [2, 3], which asked about the fixed-parameter tractability of the easier complete version of the problem (i.e., when $\lambda(M) = 0$).
4. We provide an XP-algorithm for RAD-CLUSTER-COMPLETION parameterized by $r + \lambda(M)$ in which the degree of the polynomial in the runtime has only a logarithmic dependence on r (Theorem 16). We remark that the problem is in XP parameterized by k alone (Observation 13).
5. We provide an accompanying $W[1]$ -hardness result for RAD-CLUSTER-COMPLETION that rules out its fixed-parameter tractability when parameterized by $k + r$ (Theorem 12). Since the problem is NP-hard for fixed r (as also follows from Theorem 12), this leaves only two questions open for the considered parameterizations: whether RAD-CLUSTER-COMPLETION is FPT when parameterized by either $k + \lambda(M)$ or by $r + \lambda(M)$.
6. We give an FPT-approximation scheme for the optimization version (w.r.t. the cluster size) of RAD-CLUSTER-COMPLETION.

Related Work

The RAD-CLUSTER problem (i.e., RAD-CLUSTER-COMPLETION for complete data) and variants of it were studied as early as the 1980’s, albeit under different names. Dyer and Frieze presented a heuristic algorithm for approximating a variant of RAD-CLUSTER, referred to as the p -CENTER problem, where the goal is to compute $p \in \mathbb{N}$ clusters, each of radius at most r , that contain all vectors of M ; hence, RAD-CLUSTER corresponds to the case of p -CENTER where $p = 1$ and $k = |M|$ (i.e., when the cluster contains all vectors in M). Cabello et al. [4] studied the parameterized complexity of the geometric p -CENTER PROBLEM in \mathbb{R}^d .

Frances and Litman [13] studied the complexity of RAD-CLUSTER with $k = |M|$, in the context of computing the radius of a binary code; they referred to it as the COVERING RADIUS problem and showed it to be NP-hard. Gąsieniec et al. [15, 16] studied (the optimization versions of) RAD-CLUSTER and DIAM-CLUSTER with $k = |M|$ and obtained polynomial-time algorithms as well as lower bounds for a number of cases. They also obtained 2-approximation algorithms for these problems by extending an earlier algorithm by Gonzalez [17].

The RAD-CLUSTER problem restricted to the subcase of $k = |M|$ was also extensively studied under the nomenclature CLOSEST STRING. Li et al. [24] showed that the problem admits a polynomial time approximation scheme if the goal is to minimize r . Gramm et al. [18] studied CLOSEST STRING from the parameterized complexity perspective and showed it to be fixed-parameter tractable parameterized by r . Following this naming convention, Boucher and Ma [2], and Bulteau and Schmid [3] studied the parameterized complexity of RAD-CLUSTER under the nomenclature CLOSEST STRING WITH OUTLIERS. They considered several parameters, including some of the parameters under consideration in this paper. Notably, the restriction of our fixed-parameter algorithm for RAD-CLUSTER parameterized

by $k + r + \lambda$ to the subcase where $\lambda = 0$ answers an open question in [2, see, e.g., Table 1]. Moreover, our XP algorithm for RAD-CLUSTER-COMPLETION provided in Theorem 16 that has a run-time in which the degree of the polynomial has only a logarithmic dependence on r , immediately implies an algorithm of the same running time for CLOSEST STRING WITH OUTLIERS, as a special case.

For incomplete data, Hermelin and Rozenberg [19] studied the parameterized complexity of RAD-CLUSTER-COMPLETION for $k = |M|$ under the nomenclature CLOSEST STRING WITH WILDCARDS problem, with respect to several parameterizations. Very recently, Koana et al. [20] revisited the earlier work of Hermelin and Rozenberg [19] and obtained, among other results, a fixed-parameter algorithm for the problem parameterized by r plus the maximum number of missing entries per row. Even more recently, the same group [21] also studied a problem related to DIAM-CLUSTER-COMPLETION for $k = |M|$. They obtain a classical-complexity classification w.r.t. constant lower and upper bounds on the diameter and the maximum number of missing entries per row.

2 Preliminaries

We assume basic familiarity with parameterized complexity, including the classes $W[1]$, FPT, XP, as well as Turing kernelization and FPT-approximation schemes [7, 6, 27].

Vector Terminology. Let \vec{a} and \vec{b} be two vectors in $\{0, 1, \square\}^d$, where \square is used to represent coordinates whose value is unknown (i.e., missing entries). We denote by $\Delta(\vec{a}, \vec{b})$ the set of coordinates in which \vec{a} and \vec{b} are guaranteed to differ, i.e., $\Delta(\vec{a}, \vec{b}) = \{i \mid (\vec{a}[i] = 1 \wedge \vec{b}[i] = 0) \vee (\vec{a}[i] = 0 \wedge \vec{b}[i] = 1)\}$, and we denote by $\delta(\vec{a}, \vec{b})$ the *Hamming distance* between \vec{a} and \vec{b} measured only between known entries, i.e., $|\Delta(\vec{a}, \vec{b})|$. Moreover, for a subset $D' \subseteq [d]$ of coordinates, where $[d] = \{1, \dots, d\}$, we denote by $\vec{a}[D']$ the vector \vec{a} restricted to the coordinates in D' .

There is a one-to-one correspondence between vectors in $\{0, 1\}^d$ and subsets of coordinates, i.e., for every vector, we can associate the unique subset of coordinates containing all its one-coordinates and vice-versa. It will be useful to represent a vector by the set of coordinates where the vector has the value 1. We introduce the following notation for vectors to switch between their set-representation and vector-representation. We denote by $\Delta(\vec{a})$ the set $\Delta(\vec{0}, \vec{a})$. We extend this notation to sets of vectors as follows: for a set N of vectors in $\{0, 1, \square\}^d$, we denote by $\Delta(N)$ the set $\{\Delta(\vec{v}) \mid \vec{v} \in N\}$. We say that a vector $\vec{a} \in \{0, 1, \square\}^d$ is a *t-vector* if $|\Delta(\vec{a})| = t$ and we say that \vec{a} *contains* a subset S of coordinates if $S \subseteq \Delta(\vec{a})$.

We say that a multiset¹ $M^* \subseteq \{0, 1\}^d$ is a *completion* of a multiset $M \subseteq \{0, 1, \square\}^d$ if there is a bijection $\alpha : M \rightarrow M^*$ such that for all $\vec{a} \in M$ and all $i \in [d]$ it holds that either $\vec{a}[i] = \square$ or $\alpha(\vec{a})[i] = \vec{a}[i]$. For a multiset M of vectors over $\{0, 1, \square\}^d$, we let the *deletion distance to near-completion*, $\lambda(M)$, denote the minimum integer such that there exists a subset $D_M \subseteq M$ with the following properties: (a) $|D_M| \leq \lambda(M)$, and (b) every vector in $M \setminus D_M$ contains at most $\lambda(M)$ missing entries. We call D_M the *deletion (multi-)set*, and observe that $\lambda(M)$ along with a corresponding deletion set can be trivially computed from M in linear time.

A *sunflower* in a set family \mathcal{F} is a subset $\mathcal{F}' \subseteq \mathcal{F}$ such that all pairs of elements in \mathcal{F}' have the same intersection. We will say that a multiset P is a *DIAM-Cluster* (or $|P|$ -*DIAM-Cluster*) if $\delta(\vec{p}, \vec{q}) \leq r$ for every pair $\vec{p}, \vec{q} \in P$. Similarly, P is a *RAD-Cluster* (or $|P|$ -*RAD-Cluster*) if there exists a vector $\vec{c} \in \{0, 1\}^d$ such that $\delta(\vec{c}, \vec{p}) \leq r$ for every $\vec{p} \in P$.

¹ We remark that, in the interest of brevity and when clear from context, we will sometimes use standard set notation such as $A \subseteq B$ in conjunction with multisets.

In all problems under consideration, the input size is considered to be $|M|$, i.e., the size of the matrix including multiplicities.

3 Finding a DIAM-Cluster in Incomplete Data

In this section, we present our results for DIAM-CLUSTER-COMPLETION. Our main algorithmic results are that DIAM-CLUSTER-COMPLETION is FPT parameterized by $r + \lambda(M)$ and is in XP parameterized by k alone. Together with Theorem 11 (showing the W[1]-hardness of DIAM-CLUSTER parameterized by k) and Theorem 12 (showing the W[1]-hardness of DIAM-CLUSTER-COMPLETION parameterized by k even for $r = 0$), this gives a complete complexity landscape for DIAM-CLUSTER-COMPLETION parameterized by any combination of the parameters k , r , and $\lambda(M)$.

3.1 DIAM-CLUSTER-COMPLETION Parameterized by $k + r + \lambda(M)$

We start by showing that DIAM-CLUSTER-COMPLETION parameterized by $k + r + \lambda(M)$ is FPT. We will later show a stronger result, namely that the same result already holds if we only parameterize by $r + \lambda(M)$. Showing the weaker result here is important for the following reasons: (1) we use the algorithm presented here as a subroutine in our result for the parameterization $r + \lambda(M)$, (2) the techniques developed here can also be employed for RAD-CLUSTER-COMPLETION, and (3) we obtain a Turing kernel of size polynomial in k .

The main approach behind the Turing kernel is to guess two vectors of maximum distance in the desired cluster. This will allow us to pre-process the instance such that if the resulting instance contains too many vectors, then it has a solution. Note that this approach only works for the case that a solution contains at least two vectors from $M \setminus D_M$ (recall that D_M denotes the deletion set); otherwise, we can guess the at most one vector from $M \setminus D_M$ that is in the solution and remove all the other vectors from $M \setminus D_M$. Therefore, in all cases, we end up with a reduced instance with boundedly many vectors and we will then show that we can remove all but boundedly many coordinates while preserving solutions.

► **Theorem 1.** *DIAM-CLUSTER-COMPLETION parameterized by $k + r + \lambda(M)$ has a Turing-kernel containing at most $n = k3^{2\lambda(M)+r} + \lambda(M) + 2$ vectors, each having at most $\max\{r(n-1) + \lambda(M), \binom{\lambda(M)}{2}(r+1)\}$ coordinates.*

3.2 DIAM-CLUSTER-COMPLETION Parameterized by $r + \lambda(M)$

With Theorem 1 in hand, we can move on to establishing the fixed-parameter tractability of DIAM-CLUSTER-COMPLETION parameterized by $r + \lambda(M)$. At the heart of our approach lies a new technique for analyzing the structure of vectors through sunflowers in their set representations, which we dub *iterative sunflower harvesting*. We first preprocess the instance to establish some basic properties. We then show a general result about sunflowers that allows us to derive a succinct representation of the solution cluster – in particular, this guarantees that the hypothetical solution can be described by a bounded number of sunflower cores. Finally, we proceed to “harvesting” these sunflowers cores using a branching procedure, thus computing the solution cluster.

3.2.1 Preprocessing the Instance

Let (M, k, r) be an instance of DIAM-CLUSTER-COMPLETION, let M^* be a completion of M that contains a maximum size DIAM-Cluster, and fix P^* to be such a maximum size DIAM-Cluster in M^* . We also fix D_M to be a $\lambda(M)$ -deletion set. The goal of the algorithm is to find M^* and P^* .

If $k < \lambda(M) + 2$, then we can use the algorithm in Theorem 1 to obtain a Turing kernel whose size is a function of $r + \lambda(M)$, which, in turn, would imply that DIAM-CLUSTER-COMPLETION is FPT parameterized by $r + \lambda(M)$, thus giving the desired result. We assume henceforth that P^* contains the completion of at least two vectors in $M \setminus D_M$. We can guess the subset P_R of D_M that will be completed to vectors in P^* , and restrict our attention to finding a DIAM-Cluster in $M \setminus D_M$ of size $|P^* \setminus D_M|$. We will do so by enumerating all such DIAM-Cluster's in $M \setminus D_M$ and at the end check whether one of them, together with P_R , can be extended into a DIAM-Cluster.

Therefore, we will focus on what follows on finding a cluster in $M \setminus D_M$. We first guess two vectors \vec{v} and \vec{u} in $M \setminus D_M$, together with their completions \vec{v}^* and \vec{u}^* , respectively, such that \vec{v}^* and \vec{u}^* are both in P^* and are the farthest vectors apart in $P^* \setminus D_M$; fix $r_{\max} = \delta(\vec{v}^*, \vec{u}^*)$. We remove all other vectors that can be completed into \vec{v}^* or \vec{u}^* , and reduce k accordingly; hence, we do not keep duplicates of the two vectors that we already know to be in P^* . We then normalize all vectors in M so that \vec{v}^* becomes the all-zero vector, i.e., we replace \vec{v}^* by the all-zero vector, and for every other vector $\vec{w} \neq \vec{v}$, we replace it with the vector \vec{w}' such that $\vec{w}'[i] = 0$ if $\vec{v}^*[i] = \vec{w}[i]$, $\vec{w}'[i] = \square$ if $\vec{w}[i] = \square$, and $\vec{w}'[i] = 1$, otherwise. Finally, for each vector $\vec{w} \in M \setminus D_M$, we compute the set $\Lambda(\vec{w})$ of all completions of \vec{w} at distance at most r_{\max} from both \vec{v}^* and \vec{u}^* . Note that $\Lambda(\vec{w})$ can be computed in $\mathcal{O}(2^{\lambda(M)} \cdot d)$ time for each vector in $M \setminus D_M$, where d is the dimension of the vectors in M . We then remove all vectors \vec{w} with $\Lambda(\vec{w}) = \emptyset$ from $M \setminus D_M$.

We will extend the notation $\Lambda(\vec{w})$ to $\Lambda_C(\vec{w})$, for a multiset C of vectors in $\{0, 1\}^d$, such that $\Lambda_C(\vec{w})$ is the set of all completions of a vector \vec{w} at distance at most r_{\max} to all vectors in C . We are now ready to show that after normalizing the vectors in P^* , the multiset $P^* \setminus D_M$ satisfies certain structural properties that we refer to as an r -saturated subset (of M^*); these structural properties allow for a succinct representation of P^* .

3.2.2 Sunflower Fields or Representing Sets by Cores of Sunflowers

In this subsection, we provide the central component for our algorithm based on the iterative sunflower harvesting technique. Crucial to this component is a general structural lemma that allows us to represent a family of sets in a succinct manner in terms of sunflower cores, which we believe to be interesting in its own right. We first state the result in its most general form (for sets), and then show how to adapt it to our setting.

► **Definition 2.** *Let U be a universe, \mathcal{B} a family of subsets of U and $\mathcal{A} \subseteq \mathcal{B}$. We say that \mathcal{A} is an r -saturated subfamily of \mathcal{B} (for $r \in \mathbb{N}$) if the following holds for every $t \in \mathbb{N}$ and every sunflower $S \subseteq \mathcal{A}$ containing at least $r + 1$ sets of cardinality t with core C : \mathcal{A} contains every set $B \in \mathcal{B}$ of cardinality t such that $C \subseteq B$.*

Intuitively, this property states that \mathcal{A} contains all sets in \mathcal{B} which are super-sets of cores of every sufficiently-large sunflower in \mathcal{A} (with sets of the same cardinality).

The connection of this set property to clusters is as follows. We will show that every maximal cluster is an r -saturated subset of M^* . Since P^* contains the all-zero vector \vec{v}^* , any vector in P^* contains at most r ones. Fix $t \in [r]$, and consider the set of all vectors in P^* containing exactly t ones. The above notion will allow to draw the following assumption: if the aforementioned set of vectors is large, then it must contain a large sunflower and all the vectors in M whose completions share the core of this sunflower *must be* in P . This property will subsequently allow us to represent every hypothetical solution using a bounded number of sunflowers, as we show next.

The crucial insight now is that every r -saturated subfamily containing only sets of bounded size admits a succinct representation, where we can completely describe the set via a bounded number of sunflower cores. This is made precise in the following lemma.

► **Lemma 3.** *Let \mathcal{A} and \mathcal{B} be two families of sets of cardinality at most r' over universe U such that $\mathcal{A} \subseteq \mathcal{B}$. If \mathcal{A} is an r -saturated subset of \mathcal{B} , then there is a set \mathcal{S} of at most $(rr')^{r'}$ subsets of U such that \mathcal{A} is equal to the set of all sets B in \mathcal{B} satisfying that $S \subseteq B$ for some $S \in \mathcal{S}$. Moreover, for each set $S \in \mathcal{S}$, S is either the core of a sunflower in \mathcal{A} with at least $r + 1$ petals, or $|S| = r'$.*

We will now show how Lemma 3 can be employed in our setting. Let $M, M' \subseteq \{0, 1\}^d$ with $M' \subseteq M$. Then M' is an r -saturated subset of M if $\Delta(M')$ is an r -saturated subset of $\Delta(M)$. Herein, one can think of M as being (a part of) the input matrix and M' as being an inclusion-wise maximal cluster; we will later show that this property guarantees that M' is an r -saturated subset of M . Using this definition, the following corollary now follows immediately from Lemma 3.

► **Corollary 4.** *Let $r', r \in \mathbb{N}$ and $M, M' \subseteq \{0, 1\}^d$ be sets of r' -vectors such that M' is an r -saturated subset of M . There is a set \mathcal{S} of at most $(rr')^{r'}$ subsets of $[d]$ such that M' is equal to the set of all vectors \vec{m} in M satisfying $S \subseteq \Delta(\vec{m})$ for some $S \in \mathcal{S}$. Moreover, for each set $S \in \mathcal{S}$, S is either a core of a sunflower in $\Delta(M')$ with at least $r + 1$ petals, or $|S| = r'$.*

We now can show that after normalizing the vectors in P^* , the multiset $P^* \setminus D_M$ is an r -saturated subset of M^* .

► **Lemma 5.** *Let (M, k, r) be an instance of DIAM-CLUSTER-COMPLETION, let M^* be a completion of M and let P^* be a DIAM-Cluster in M^* of maximum size such that $\vec{0} \in P^*$. Then for every $N \subseteq M^*$, $P^* \setminus N$ is an r -saturated subset of $M^* \setminus N$.*

Since $P^* \setminus N$ is an r -saturated subset of $M^* \setminus N$, by Corollary 4, applied separately for each $r' \in [r]$, there exists a set $\mathcal{S} = \{(S_1, r_1), \dots, (S_\ell, r_\ell)\}$, with $\ell \leq \sum_{r' \in [r]} (rr')^{r'} \leq r^{2r+1}$, such that $P^* \setminus N$ contains precisely all the vectors \vec{w} in M^* , such that for some (S_i, r_i) , $i \in [\ell]$, $S_i \subseteq \Delta(\vec{w})$ and $|\Delta(\vec{w})| = r_i$.

We call the pair (S_i, r_i) an r_i -center (of $P^* \setminus N$ in $M^* \setminus N$). We say that a vector $\vec{w} \in \{0, 1, \square\}^d$ is *compatible* with r_i -center (S_i, r_i) if there is a completion $\vec{w}^* \in \{0, 1\}^d$ of \vec{w} , called *witness of compatibility*, such that $S_i \subseteq \Delta(\vec{w}^*)$ and $|\Delta(\vec{w}^*)| = r_i$. We say that $\vec{w} \in \{0, 1, \square\}^d$ is *compatible* with \mathcal{S} if it is compatible with some $(S_i, r_i) \in \mathcal{S}$.

The *size* of an r_i -center is the number of vectors that are compatible with it in M . Moreover, for a set $\mathcal{S} = \{(S_1, r_1), \dots, (S_\ell, r_\ell)\}$ and a multiset C of vectors from $\{0, 1\}^d$, we say that \mathcal{S} *defines* C , if every vector $\vec{c} \in C$ is compatible with \mathcal{S} and for every $(S_i, r_i) \in \mathcal{S}$ there is a vector in C compatible with (S_i, r_i) . We say that \mathcal{S} *properly defines* C , if $|\mathcal{S}| \leq r^{2r+1}$, \mathcal{S} defines C , and for every $(S_i, r_i) \in \mathcal{S}$ either:

- $|S_i| = r_i$ and the unique vector that is compatible with (S_i, r_i) is in C ; or
- $|S_i| < r_i$ and C contains a set N of $r + 1$ r_i -vectors such that $\Delta(N)$ forms a sunflower with core S_i .

Note that if every vector in C has at most r_{\max} 1's, then since C is an r -saturated subset of C , it follows from Corollary 4 that there always exists a set \mathcal{S} that properly defines C .

► **Observation 6.** *Let C be a multiset of vectors from $\{0, 1\}^d$, with $\max_{\vec{c} \in C} |\Delta(\vec{c})| \leq r$. Then there exists a set \mathcal{S} of at most r^{2r+1} r_i -centers that properly defines C .*

Suppose that we have a correct guess for \mathcal{S} , then we can already solve the problem as follows. For each $(S_i, r_i) \in \mathcal{S}$ such that $|S_i| = r_i$, there is only one possible r_i -vector that contains S_i . If our guess is correct, then P^* contains at least one vector that can be completed to this particular r_i -vector, and by maximality of P^* , P^* has to contain all such vectors. If $(S_i, r_i) \in \mathcal{S}$ such that $|S_i| < r_i$, then P^* contains a sunflower containing r_i -vectors of size at least $r + 1$ whose core is S_i . Clearly, P^* contains all the vectors that can be completed to an r_i -vector containing S_i , since Lemma 5 holds for any completion M^* of M that contains P^* as a subset. The following lemma shows that all such vectors can be completed arbitrarily since all that matters is that their completion is compatible with \mathcal{S} .

► **Lemma 7.** *Let (M, k, r) be an instance of DIAM-CLUSTER-COMPLETION, M^* a completion of M , P^* a DIAM-Cluster in M^* of maximum size with $\vec{0} \in P^*$, and $N \subseteq M^*$. If \mathcal{S} properly defines $P^* \setminus N$, then for every pair of vectors $\vec{w}_1, \vec{w}_2 \in \{0, 1\}^d$ compatible with \mathcal{S} it holds that $\delta(\vec{w}_1, \vec{w}_2) \leq r$.*

It is easy to see that there are at most d^{r+1} choices for a pair (S_i, r_i) (i.e., a r_i -center), and hence we have at most $(d^{r+1})^{r^{2r+1}} = d^{\mathcal{O}(r^{2r+2})}$ possible choices for the set \mathcal{S} that properly defines P^* . This bound already implies an XP-algorithm. To obtain a fixed-parameter algorithm, it suffices to find the correct guess for \mathcal{S} in FPT-time, which is our next goal.

3.2.3 Iterative Sunflower Harvesting

We are now ready to describe the iterative sunflower harvesting procedure, which allows us to obtain the desired FPT-algorithm. Namely, we show that instead of enumerating all $d^{r^{2r+2}}$ possible sets \mathcal{S} of r_i -centers to find the one that properly defines $P^* \setminus D_M$, it suffices to enumerate only $f(r, \lambda(M))$ -many “important” r_i -centers for each choice of \vec{v}^* and \vec{u}^* (recall that \vec{v}^* and \vec{u}^* are the two fixed vectors in $P^* \setminus D_M$ that were guessed), where f is some function that depends only on r and $\lambda(M)$. Moreover, we can enumerate these possibilities in FPT-time.

We compute \mathcal{S} by iteratively adding r_i -centers one by one. The main idea is to show that, for any partial solution \mathcal{S}' , there is a bounded number of choices for the next r_i -center to add. As a first step in this direction, the following lemma shows that for \mathcal{S}' , there is always a “large” r_i -center (S_i, r_i) that can be added to \mathcal{S}' , i.e., of size at least a $(2^r r^{2r+1})$ -fraction of the remaining vectors. Before we state the lemma, we introduce the following notations. If \vec{w} is compatible with \mathcal{S} , we will denote by $\zeta^{\mathcal{S}}(\vec{w})$ the set of witnesses of compatibility for \vec{w} and \mathcal{S} . Recall that, for a vector $\vec{w} \in \{0, 1, \square\}^d$ and multiset C of vectors from $\{0, 1\}^d$, $\Lambda_C(\vec{w})$ denotes the set of all completions of vector \vec{w} at distance at most r_{\max} to all vectors in C , i.e., $\max_{\vec{c} \in C} \{\delta(\vec{c}, \vec{c}_w)\} \leq r_{\max}$.

► **Lemma 8.** *Let P^* be a maximum DIAM-Cluster in (M, k, r) , \mathcal{S} the set of r_i -centers that properly define $P^* \setminus D_M$, and $\mathcal{S}' \subseteq \mathcal{S}$. Moreover, let C' be the multiset of vectors \vec{w} in $M \setminus D_M$ with $\zeta^{\mathcal{S}'}(\vec{w}) \neq \emptyset$ and C the multiset containing a vector $\vec{w}_c \in \zeta^{\mathcal{S}'}(\vec{w})$ for every $\vec{w} \in C'$. Finally, let M' be the multiset consisting of all the vectors $\vec{w} \in M \setminus (C \cup D_M)$ with $\Lambda_C(\vec{w}) \neq \emptyset$. Then there exists $(S_i, r_i) \in \mathcal{S} \setminus \mathcal{S}'$ such that at least $(|M'|/2^{r_{\max}} - |D_M|)/r^{2r+1}$ vectors in M' are compatible with (S_i, r_i) .*

Note that each normalised vector \vec{w} can be compatible with at most $2^{r+\lambda(M)}$ r_i -centers (S_i, r_i) , since $S_i \subseteq \Delta(\vec{w}^*)$ for some completion \vec{w}^* of \vec{w} . Now it follows from a counting argument that the number of large r_i -centers is at most $2^{r+\lambda(M)}(2^r r^{2r+1}) = 2^{2r+\lambda(M)} r^{2r+1}$ and those can be enumerated in time $\mathcal{O}(2^{r+\lambda(M)} |M|)$. By Observation 6, $|\mathcal{S}|$ and hence the depth of the branching algorithm, is at most r^{2r+1} , which implies the following theorem.

► **Theorem 9.** *DIAM-CLUSTER-COMPLETION is fixed-parameter tractable parameterized by $r + \lambda(M)$.*

3.3 DIAM-CLUSTER-COMPLETION Parameterized by k

Here, we use an Integer Linear Programming subroutine to show that DIAM-CLUSTER-COMPLETION parameterized by k is in XP.

Moreover, we also observe in Theorem 11 that, unless $W[1]=FPT$, this cannot be improved to an FPT-algorithm even for complete data.

► **Theorem 10.** DIAM-CLUSTER-COMPLETION is in XP parameterized by k .

Proof. Let (M, k, r) be an instance of DIAM-CLUSTER-COMPLETION. The algorithm works by enumerating all potential clusters C of size exactly k , and then uses a reduction to an ILP instance with $f(k)$ variables to check whether C can be completed into a cluster. Since there are at most $|M|^k$ many potential clusters of size exactly k , it only remains to show how to decide whether a given set C of exactly k vectors in M can be completed into a DIAM-Cluster. Let M_C be the submatrix of M containing only the vectors in C . Then M_C has at most 3^k distinct columns, and moreover, each of those columns can be completed in at most 2^k possible ways. Let T be the set of all columns occurring in M_C and for a column $\vec{t} \in T$, let $F(\vec{t})$ be the set of all possible completions of \vec{t} , and let $\#(\vec{t})$ denote the number of columns in M_C equal to \vec{t} . For a vector $\vec{f} \in \{0, 1\}^k$ (representing the completion of a column), let $T(\vec{f})$ denote the subset of T containing all columns \vec{t} with $\vec{f} \in F(\vec{t})$. Moreover, for every i and j with $1 \leq i < j \leq k$ (representing the i -th and the j -th vectors in C), we denote by $FD(i, j)$ the set of all vectors (completions of columns) $\vec{f} \in \{0, 1\}^k$ such that $\vec{f}[i] \neq \vec{f}[j]$.

We are now ready to construct an ILP instance \mathcal{I} with at most $3^k 2^k$ variables that is feasible if and only if C can be completed into a DIAM-Cluster. \mathcal{I} has one variable $x_{\vec{t}, \vec{f}}$ for every $\vec{t} \in T$ and every $\vec{f} \in F(\vec{t})$ whose value (in a feasible assignment) represents how many columns of type \vec{t} in M_C will be completed to \vec{f} . Moreover, \mathcal{I} has the following constraints:

- One constraint for every $\vec{t} \in T$ stipulating that every column of type \vec{t} in M_C is completed in some manner:

$$\sum_{\vec{f} \in F(\vec{t})} x_{\vec{t}, \vec{f}} = \#(\vec{t}).$$

- For every i and j with $1 \leq i < j \leq k$ (representing the i -th and the j -th vectors in C), one constraint stipulating that the Hamming distance between the i -th and the j -th vectors in C does not exceed r :

$$\sum_{\vec{f} \in FD(i, j)} x_{\vec{t}, \vec{f}} \leq r.$$

This completes the construction of \mathcal{I} and it is straightforward to verify that \mathcal{I} has a feasible assignment if and only if C can be completed to a DIAM-Cluster. Since \mathcal{I} has at most $3^k 2^k$ variables, and since it is well known that ILP can be solved in FPT-time w.r.t. the number of variables [23], \mathcal{I} can be solved in FPT-time w.r.t. k . ◀

► **Theorem 11.** DIAM-CLUSTER-COMPLETION is $W[1]$ -hard parameterized by k even if $\lambda(M) = 0$.

We note that our second result also establishes the $W[1]$ -hardness of RAD-CLUSTER-COMPLETION (since both problems coincide when $r = 0$).

► **Theorem 12.** DIAM-CLUSTER-COMPLETION and RAD-CLUSTER-COMPLETION are both $W[1]$ -hard parameterized by k even if $r = 0$.

4 Finding a RAD-Cluster in Incomplete Data

In this section, we present our results for RAD-CLUSTER-COMPLETION. We will show that RAD-CLUSTER-COMPLETION is FPT parameterized by $k + r + \lambda(M)$, and is in XP parameterized by $r + \lambda$ alone. Notably, the degree of the polynomial in the run-time of our XP algorithm grows only logarithmically in r and the algorithm can be employed to solve the CLOSEST STRING WITH OUTLIERS problem [2, 3].

Before proceeding to the main contributions of this section, we observe that, by combining the trivial branching procedure, in which we branch over all sets of k vectors from M (where in each branch we proceed under the assumption that all vectors outside of the set can be deleted), with a previous result of Hermelin and Rozenberg [19, Theorem 2], which solves the special case of RAD-CLUSTER-COMPLETION for $k = |M|$, we obtain:

► **Observation 13.** RAD-CLUSTER-COMPLETION parameterized by k is in XP.

Together with the previously-established Theorem 12 (showing the W[1]-hardness for RAD-CLUSTER-COMPLETION w.r.t. k even for $r = 0$), this gives us an almost complete picture of the parameterized complexity of RAD-CLUSTER-COMPLETION for any combination of the parameters $k, r, \lambda(M)$. The only two questions that remain open are whether the XP result for $r + \lambda(M)$ can be improved to an FPT-result (as this has been the case for DIAM-CLUSTER-COMPLETION), and whether it is possible to obtain an FPT-algorithm either for parameter k or $k + \lambda(M)$. As a first step in this direction, we present an FPT-approximation scheme for parameter $r + \lambda(M)$ in Section 4.3. The following observation will be useful:

► **Observation 14.** Given a (complete) vector $\vec{s} \in \{0, 1\}^d$, in time $\mathcal{O}(|M|d)$ we can decide if \vec{s} is the center of a RAD-CLUSTER of k vectors in M .

Observation 14 is straightforward since we can find all vectors $\vec{w} \in M$ that can be completed to a vector \vec{w}^* at distance at most r from \vec{s} by letting $\vec{w}^*[i] = \vec{s}[i]$ wherever $\vec{w}^*[i] = \square$, and then decide whether \vec{w}^* is such a vector by computing $\delta(\vec{w}^*, \vec{s})$.

4.1 RAD-CLUSTER-COMPLETION Parameterized by $k + r + \lambda(M)$

We start by showing that, as in the case of DIAM-CLUSTER-COMPLETION, RAD-CLUSTER-COMPLETION parameterized by $k + r + \lambda(M)$ has a Turing kernel. The approach is similar to that in Subsection 3.1.

► **Theorem 15.** RAD-CLUSTER-COMPLETION parameterized by $k + r + \lambda(M)$ has a Turing-kernel containing at most $n = k3^{\lambda(M)+2r} + \lambda(M) + 2$ vectors, each having at most $\max\{2r(n-1) + \lambda(M), \binom{\lambda(M)}{2}(2r+1)\}$ coordinates.

4.2 RAD-CLUSTER-COMPLETION Parameterized by $r + \lambda(M)$

While, it is relatively easy to see that RAD-CLUSTER-COMPLETION parameterized by $r + \lambda(M)$ can be solved in time $f(\lambda(M), r)n^{\mathcal{O}(r)}$, here we provide a more efficient algorithm by reducing the degree of the polynomial in the run-time from $\mathcal{O}(r)$ to $\log r$. Moreover, our algorithm can be applied to the CLOSEST STRING WITH OUTLIERS problem [3].

► **Theorem 16.** RAD-CLUSTER-COMPLETION can be solved in time $\mathcal{O}(|M|2^{\lambda(M)}(|M|(2^{2r} + d))^{\log r + 1})$ and is therefore in XP parameterized $r + \lambda(M)$.

Proof Sketch. The main ideas behind the algorithm are captured by the following definition and discussions. Let $F \subseteq [d]$ and let t be an integer, where $0 \leq t \leq r$. We say that a vector $\vec{v} \in \{0, 1\}^d$ is an (F, t) -seed for a center $\vec{c} \in \{0, 1\}^d$ of a solution for (M, k, r) if it satisfies:

- (C1) \vec{c} agrees with \vec{v} on all coordinates in F ; and
 (C2) \vec{c} differs from \vec{v} on at most t coordinates outside of F .

We can show the following statement. If \vec{v} is an (F, t) -seed for \vec{c} , then either \vec{v} is the center of a solution for (M, k, r) , or there is a vector $\vec{m} \in M$ with $r < \delta(\vec{v}, \vec{m}) \leq 2r$ and a subset $C \subset D \setminus F$, where $D = \Delta(\vec{v}, \vec{m})$, such that the vector \vec{v}' obtained from \vec{v} by complementing all coordinates in C is an $(F \cup D, t/2)$ -seed for \vec{c} . Note that testing the former possibility, that is, whether a vector $\vec{v} \in \{0, 1\}^d$ is a center of a solution for (M, k, r) , can be done in time $\mathcal{O}(|M|d)$ by Observation 14.

Since there are at most $M2^{2r}$ possibilities for \vec{m} and C , we can use the above statement to obtain a $(F \cup D, t/2)$ -seed from a given (F, t) -seed. This can be employed within a recursive procedure that, given a (\emptyset, r) -seed for some center \vec{c} of a solution either obtains a center of a solution or obtains a $(F', 0)$ -seed (which itself is the center of a solution), in at most $\log r$ recursive steps. It only remains to find a (\emptyset, r) -seed for some center \vec{c} of a solution, which can be achieved by guessing the completion \vec{v} of any vector in $M \setminus D_M$ that will be in a solution. Note that if $M \setminus D_M$ does not contain a vector in the solution, then $k \leq \lambda(M)$ and the result follows from Theorem 15. \blacktriangleleft

We note that the algorithm provided by the above theorem generalizes a previous algorithm of Marx [26, Lemma 3.2] for CLOSEST SUBSTRING to strings that may contain unknown characters. In particular, it lifts the concept of “generators” to strings with unknown characters by showing that there are $\log r$ vectors that can be computed efficiently and that define at most $r \log r$ “important” coordinates for the center of some solution.

4.3 FPT Approximation Scheme Parameterized by $r + \lambda(M)$

In this subsection we give an algorithm that, for a given instance (M, k, r) of RAD-CLUSTER-COMPLETION and $\varepsilon \in \mathbb{R}$, where $0 < \varepsilon < 1$, computes in FPT-time parameterized by $r + \lambda(M) + \frac{1}{\varepsilon}$ a center of a RAD-cluster of size at least $(1 - \varepsilon)k$, or it correctly concludes that no RAD-Cluster of size k exists.

► **Theorem 17.** *Given an instance (M, k, r) of RAD-CLUSTER-COMPLETION and $\varepsilon \in \mathbb{R}$, where $0 < \varepsilon < 1$, there exists an FPT algorithm \mathcal{A} , parameterized by $r + \lambda(M) + \frac{1}{\varepsilon}$, such that \mathcal{A} either computes a RAD-Cluster of size at least $(1 - \varepsilon)k$, or correctly concludes that M does not contain a RAD-Cluster of size k .*

Proof Sketch. The algorithm starts by performing a similar branching and pre-processing to the FPT algorithm for DIAM-CLUSTER-COMPLETION parameterized by $r + \lambda(M)$. Fix M^* to be a completion of M that contains a maximum size RAD-Cluster, let P^* be such a maximum size RAD-Cluster in M^* , and let \vec{s}^* be a center of P^* . The goal of the algorithm is to find \vec{s}^* , as given \vec{s}^* , by Observation 14, we can decide the instance in time $\mathcal{O}(|M|d)$.

If $k < \frac{2\lambda(M)}{\varepsilon} + 2$, then we use the algorithm in Theorem 15 to obtain a Turing kernel. Otherwise, we can guess two vectors \vec{u} and \vec{v} in $M \setminus D_M$, together with their respective completions \vec{u}^* and \vec{v}^* , such that \vec{u}^* and \vec{v}^* are the farthest vectors apart in $P^* \setminus D_M$; fix $r_{\max} = \delta(\vec{v}^*, \vec{u}^*)$. We normalize all the vectors in M so that \vec{v}^* becomes the all-zero vector. Finally, for each vector $\vec{w} \in M$, we can in time $r_{\max} \cdot d$, where d is the dimension of the vectors in M , check if there is a completion \vec{w}^* of \vec{w} such that the distance from \vec{w}^* to both \vec{v}^* and \vec{u}^* is at most r_{\max} ; we remove all vectors \vec{w} that do not have such a completion. Note here that some vectors in $P^* \cap D_M$ could have been removed from M at this step. However, we did not remove any vector from $P^* \setminus D_M$. Hence, after the preprocessing, M contains a RAD-Cluster with center \vec{s}^* and at least $k' = (1 - \frac{\varepsilon}{2})k$ vectors. Our goal is to

find a center for a RAD-Cluster with at least $(1 - \frac{\varepsilon}{2})k' = (1 - \frac{\varepsilon}{2})^2 k \geq (1 - \varepsilon)k$ vectors. For ease of exposition, we let $\varepsilon' = \frac{\varepsilon}{2}$ and we let $k' \geq (1 - \varepsilon')k$ be the number of vectors of P^* still in M . Now we can show the following statement: After the above pre-processing, in time $\mathcal{O}(2^{r_{\max}} \cdot |M|)$, we can find a center for a RAD-Cluster of size $\frac{|M|}{2^{r_{\max}}}$.

Therefore, we can assume henceforth that $k' \geq \frac{|M|}{2^{r_{\max}}}$. Now the algorithm sets $\vec{s}_0^* = \vec{v}^* = \vec{0}$ and the goal is to iteratively compute $\vec{s}_1^*, \vec{s}_2^*, \vec{s}_3^*, \dots, \vec{s}_{r'}^*$, $r' \leq r$, such that:

1. for all $i \in [r']$, we have $\Delta(\vec{s}_i^*) = \Delta(\vec{s}_{i-1}^*) \cup \{c_i\}$ for some coordinate $c_i \in \Delta(\vec{s}^*) \setminus \Delta(\vec{s}_{i-1}^*)$;
2. for all $j \in [r' - 1]$, the number of vectors \vec{w} with $\delta(\vec{w}, \vec{s}_j^*) \leq r$ is less than $(1 - \varepsilon')k'$; and
3. the number of vectors \vec{w} with $\delta(\vec{w}, \vec{s}_{r'}^*) \leq r$ is at least $(1 - \varepsilon')k'$.

Let \vec{s}_i^* be such that $\Delta(\vec{s}_i^*) \subseteq \Delta(\vec{s}^*)$ for some $i \in [r' - 1]$. The number of vectors at distance at most r from \vec{s}_i^* , $i < r'$, is less than $(1 - \varepsilon')k'$. This means that at least $\varepsilon'k' \geq \frac{\varepsilon'|M|}{2^{r_{\max}}}$ vectors whose completions are in P^* are at distance at least $r + 1$ from \vec{s}_i^* . For every such vector \vec{w} , it is easy to see that, since $\Delta(\vec{s}_i^*) \subseteq \Delta(\vec{s}^*)$, it must be the case that $(\Delta(\vec{s}^*) \cap \Delta(\vec{w})) \setminus \Delta(\vec{s}_i^*)$ is nonempty. Note that $|\Delta(\vec{s}^*)| \leq r$, and hence there exists $c_{i+1} \in \Delta(\vec{s}^*)$ such that, for at least $\frac{\varepsilon'|M|}{2^{r_{\max} \cdot r}}$ vectors \vec{w} in M at distance at least $r + 1$ from \vec{s}_i^* , it holds that $c_{i+1} \in \Delta(\vec{w})$. Moreover, for every vector $\vec{w} \in M$, we have $|\Delta(\vec{w})| \leq r_{\max}$. It follows—by a straightforward counting argument—that there are at most $\frac{2^{r_{\max} \cdot r}}{\varepsilon} \cdot r_{\max}$ coordinates $c \in [d]$ such that, for at least $\frac{\varepsilon'|M|}{2^{r_{\max} \cdot r}}$ vectors \vec{w} , it holds that $c \in \Delta(\vec{w})$. Therefore, to obtain \vec{s}_{i+1}^* such that $\Delta(\vec{s}_{i+1}^*) \subseteq \Delta(\vec{s}^*)$, we only need to branch on one of at most $\frac{2^{r_{\max} \cdot r}}{\varepsilon} \cdot r_{\max}$ coordinates. The statement of the theorem follows since we can exhaustively branch on the coordinates that are set to 1 in at least $\frac{\varepsilon'|M|}{2^{r_{\max} \cdot r}}$ many vectors in M , until either the number of vectors at distance at most r from \vec{s}_i^* is at least $(1 - \varepsilon')k'$ or $i \geq r$. ◀

5 Concluding Remarks

We studied the parameterized complexity of two fundamental problems pertaining to incomplete data that have applications in data analytics. In most cases, we were able to provide a complete landscape of the parameterized complexity of the problems w.r.t. the parameters under consideration. It is worth noting that all algorithmic upper bounds obtained in this paper can also be directly generalized to vectors (i.e., matrices) over a domain whose size is bounded by the parameter value by using the encoding described by Eiben et al. [9].

Two important open questions ensue from our work, namely determining the parameterized complexity of RAD-CLUSTER-COMPLETION w.r.t. each of the two parameterizations $k + \lambda$ and $r + \lambda$. In particular, the restrictions of these two problems to complete data (i.e., $\lambda = 0$) remain open, resulting in two important questions about the parameterized complexity of RAD-CLUSTER parameterized by the cluster size k or the cluster radius r .

References

- 1 Laura Balzano, Arthur Szlam, Benjamin Recht, and Robert D. Nowak. k -subspaces with missing data. *2012 IEEE Statistical Signal Processing Workshop (SSP)*, pages 612–615, 2012.
- 2 Christina Boucher and Bin Ma. Closest string with outliers. *BMC Bioinformatics*, 12(S-1):S55, 2011.
- 3 Laurent Bulteau and Markus L. Schmid. Consensus strings with small maximum distance and small distance sum. *Algorithmica*, 82(5):1378–1409, 2020.
- 4 Sergio Cabello, Panos Giannopoulos, Christian Knauer, Dániel Marx, and Günter Rote. Geometric clustering: Fixed-parameter tractability and lower bounds with respect to the dimension. *ACM Trans. Algorithms*, 7(4):43:1–43:27, 2011.

- 5 Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *Journal of Computer and System Sciences*, 68(2):417–441, 2004.
- 6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 7 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 8 M.E Dyer and A.M Frieze. A simple heuristic for the p -centre problem. *Oper. Res. Lett.*, 3(6):285–288, 1985.
- 9 Eduard Eiben, Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. The parameterized complexity of clustering incomplete data. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 7296–7304. AAAI Press, 2021.
- 10 Ehsan Elhamifar. High-rank matrix completion and clustering under self-expressive models. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 73–81. Curran Associates, Inc., 2016.
- 11 Ehsan Elhamifar and René Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(11):2765–2781, 2013.
- 12 Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC '88*, pages 434–444. ACM, 1988.
- 13 M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30(2):113–119, 1997.
- 14 Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. Parameterized algorithms for the matrix completion problem. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 1642–1651, 2018.
- 15 Leszek Gąsieniec, Jesper Jansson, and Andrzej Lingas. Efficient approximation algorithms for the Hamming center problem. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 905–906, 1999.
- 16 Leszek Gąsieniec, Jesper Jansson, and Andrzej Lingas. Approximation algorithms for Hamming clustering problems. *Journal of Discrete Algorithms*, 2(2):289–301, 2004.
- 17 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- 18 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for CLOSEST STRING and related problems. *Algorithmica*, 37(1):25–42, 2003.
- 19 Danny Hermelin and Liat Rozenberg. Parameterized complexity analysis for the closest string with wildcards problem. *Theoretical Computer Science*, 600:11–18, 2015.
- 20 Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. Parameterized algorithms for matrix completion with radius constraints. 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 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 21 Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. Binary matrix completion under diameter constraints. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPICs*, pages 47:1–47:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 22 Stefan Kratsch, Dániel Marx, and Magnus Wahlström. Parameterized complexity and kernelizability of max ones and exact ones problems. *TOCT*, 8(1):1:1–1:28, 2016.
- 23 H. W. Lenstra and Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.
- 24 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *J. ACM*, 49(2):157–171, 2002.

47:14 Finding a Cluster in Incomplete Data

- 25 Dániel Marx. Parameterized complexity of constraint satisfaction problems. *Computational Complexity*, 14(2):153–183, 2005.
- 26 Dániel Marx. Closest substring problems with small distances. *SIAM J. Comput.*, 38(4):1382–1410, 2008.
- 27 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.
- 28 J. Yi, T. Yang, R. Jin, A. K. Jain, and M. Mahdavi. Robust ensemble clustering by matrix completion. In *2012 IEEE 12th International Conference on Data Mining*, pages 1176–1181, 2012.

Lyndon Arrays Simplified

Jonas Ellert  

Department of Computer Science, Technische Universität Dortmund, Germany

Abstract

A Lyndon word is a string that is lexicographically smaller than all of its proper suffixes (e.g., **airbus** is a Lyndon word; **amtrak** is not a Lyndon word because its suffix **ak** is lexicographically smaller than **amtrak**). The Lyndon array (sometimes called Lyndon table) identifies the longest Lyndon prefix of each suffix of a string. It is well known that the Lyndon array of a length- n string can be computed in $\mathcal{O}(n)$ time. However, most of the existing algorithms require the suffix array, which has theoretical and practical disadvantages. The only known algorithms that compute the Lyndon array in $\mathcal{O}(n)$ time without the suffix array (or similar data structures) do so in a particularly space efficient way (Bille et al., ICALP 2020), or in an online manner (Badkobeh et al., CPM 2022). Due to the additional goals of space efficiency and online computation, these algorithms are complicated in technical detail. Using the main ideas of the aforementioned algorithms, we provide a simpler and easier to understand algorithm that computes the Lyndon array in $\mathcal{O}(n)$ time.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Lyndon table, Lyndon array, Lyndon word, nearest smaller suffixes, lexicographical ordering, general ordered alphabets, combinatorial algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.48

Supplementary Material *Software (Source Code)*: <https://github.com/jonas-ellert/simple-lyndon>; archived at [swh:1:dir:dbbf2b4ac2fa652eb0865cdc6719924ce8a81952](https://swh.io/1/dir/dbbf2b4ac2fa652eb0865cdc6719924ce8a81952)

1 Related Work

A Lyndon word is a string that is lexicographically smaller than all of its proper suffixes (e.g., **airbus** is a Lyndon word; **amtrak** is not a Lyndon word because its suffix **ak** is lexicographically smaller than **amtrak**). The Lyndon array (sometimes called Lyndon table) identifies the longest Lyndon prefix of each suffix of a string (a precise definition follows later). It has both theoretical and practical applications related to repetitiveness in strings. Most notably, it is a crucial component for showing that a length- n string contains less than n maximal repetitions (the “Runs” theorem by Bannai et al. [3]), and it is useful for computing all of these maximal repetitions in optimal time [8]. Other applications of the Lyndon array include text compression and indexing [16].

There is a close relation [12] between the Lyndon array and the suffix array (one of the most fundamental data structures in string algorithmics [15]). This inspired research focused on computing the Lyndon array from the suffix array [9], computing both arrays simultaneously [2, 13], and also using properties of the Lyndon array to compute the suffix array [2, 4]. One of the conceptually simplest methods for computing the Lyndon array combines the (inverse) suffix array with a folklore algorithm for the computation of nearest smaller values [9, Algorithm ISA-NSV] (we will discuss this algorithm in Section 4).

However, using the suffix array is both a theoretical and practical drawback. From a theoretical point of view, computing the suffix array of a length- n string over a general ordered alphabet takes $\Omega(n \lg n)$ time. This is due to the well-known information-theoretic lower bound on the number of comparisons for sorting. We can only compute the suffix array in optimal $\mathcal{O}(n)$ time, if the alphabet can be sorted in $\mathcal{O}(n)$ time (e.g., a polynomial integer alphabet $\Sigma = \{1, \dots, n^{\mathcal{O}(1)}\}$ on a word RAM of width $w \geq \log_2 n$). This is not a concern



© Jonas Ellert;

licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 48; pp. 48:1–48:14

Leibniz International Proceedings in Informatics



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

in practice, where this requirement is virtually always met. Nevertheless, even the fastest algorithms for the suffix array are relatively slow in practice, and the computation appears to be somewhat excessive for the seemingly simpler task of computing the Lyndon array.

There are two known algorithms that compute the Lyndon array by mere symbol comparisons, without using the suffix array, and in optimal $\mathcal{O}(n)$ time. The first one [5] computes the Lyndon array using only $\mathcal{O}(1)$ words of additional working space, or the succinct $2n$ -bit version of the Lyndon array using only $o(n)$ bits of additional working space. The second one [1] is an online algorithm that computes the Lyndon array from right to left. The technical details of these algorithms are rather intricate, and many of the used techniques are only needed due to the goals of space efficiency and online computation.

Contributions. We present a simple $\mathcal{O}(n)$ time and space algorithm that computes the Lyndon array of a length- n string in the comparison model (i.e., the only elementary operations allowed on symbols of the string are comparisons of the form less-greater-equal). It requires no precomputed data structures (like the suffix array), and works by exploiting the powerful combinatorial properties of Lyndon words. Many of the used techniques are simpler versions of what was done in [5, 1]. The final trick used to achieve linear time is similar to Manacher’s classic algorithm for longest palindromic substrings [14].

The proof of correctness is still rather technical, but the resulting algorithm is incredibly easy to implement. The provided C++ implementation consists of not even 50 lines of code, and uses neither external nor standard libraries.

2 Preliminaries

We use the interval notations $[i, j] = [i, j + 1) = (i - 1, j] = (i - 1, j + 1)$ to denote the integer set $\{i, i + 1, \dots, j\}$ (or \emptyset if $i > j$). A *string* $x = x[1..n]$ over an *ordered alphabet* Σ is a sequence $x = x[1]x[2] \cdots x[n]$ of *symbols* drawn from some totally ordered set Σ . We write $|x| = n$ to denote the length of the string. The set Σ^* contains all strings over Σ , including the *empty string* ϵ of length 0. The concatenation of two strings $x[1..n]$ and $y[1..m]$ is the sequence $x[1] \dots x[n]y[1] \dots y[m]$, and simply written as xy (or $x \cdot y$). The total order of symbols from Σ induces the *lexicographical order* of strings from Σ^* . We say that x is *lexicographically smaller* than y and write $x \prec y$, if and only if either $y = xw$ for some non-empty string w , or $x = urv$ and $y = usw$ for (possibly empty) strings u, v, w and symbols $r < s$. We write $x \preceq y$ to denote $x = y \vee x \prec y$.

For $i, j \in [1, n]$ with $i \leq j$, the *substring* $x[i..j] = x[i..j + 1) = x(i - 1..j] = x(i - 1..j + 1)$ of $x[1..n]$ is the sequence $x[i]x[i + 1] \cdots x[j]$. If $i > j$, then $x[i..j]$ equals the *empty string* denoted by ϵ . If $x[i..j] \neq \epsilon$ then $x[i..j]$ is a *proper* substring. A substring of the form $x[1..j]$ is called *prefix* of x , while $x[i..n]$ is called *suffix* of x . We use the simplified notation $x_i = x[i..n]$ for the suffix starting at position i . The *longest common extension (LCE)* of two suffixes x_i and x_j is defined as the length of the longest common prefix of the suffixes, formally $\text{LCE}(i, j) = \max\{|u| \mid u, v, w \in \Sigma^* \wedge x_i = uv \wedge x_j = uw\}$.

Sentinel Symbols. Throughout this work, we often compare two suffixes x_i, x_j of the same string $x[1..n]$. The special case where one suffix is a prefix of another, e.g., $i < j$ and $x_i = x_jx_{i+|x_j|}$, often complicates the notation of definitions and algorithms. This can be avoided by assuming that the text starts and ends with special *sentinel symbols* $x[1] = \#$ and $x[n] = \$$. The sentinels are smaller than all other symbols, i.e., $\forall k \in (1, n) : x[k] > \$ > \#$. In definitions and lemmas we emphasize the presence of sentinels by writing $x = x[1..n] = \#x(1..n)\$$. (Note

that n is the length of the string *including* the sentinels.) The usage of sentinels is of purely cosmetic nature. Particularly, they do not affect the lexicographical order of suffixes, i.e., it holds $x[i..n]\$ \prec x[j..n]\$$ if and only if $x[i..n] \prec x[j..n]$. In practice, we can add sentinels to any string by either physically prepending and appending them, or by using an appropriate wrapper function¹ when accessing the string.

Lyndon Words and Arrays

There are multiple equivalent definitions of Lyndon words. We use Duval's characterization based on the lexicographical order of suffixes:

► **Definition 1** ([7, Proposition 1.2]). *A string $x[1..n]$ is a Lyndon word, if and only if it is lexicographically smaller than all of its proper non-empty suffixes, i.e., $\forall i \in [2, n] : x \prec x_i$.*

The Lyndon array of a string and its close relatives, the nearest smaller suffix arrays, capture the combinatorial structure of Lyndon substrings. We denote the Lyndon array by λ , which was also done in [5, 9]. Other notations are *Lynd* in [6, 1], l in [3], and \mathcal{L} in [10, 11].

► **Definition 2.** *The Lyndon array $\lambda[1..n]$, the previous smaller suffix array $\text{prev}[1..n]$, and the next smaller suffix array $\text{next}[1..n]$ of a string $x = \#x(1..n)\$$ are defined as:*

- (i) $\forall i \in [1, n] : \lambda[i] = \max\{m \mid m \in [1, n - i + 1] \wedge x[i..i + m] \text{ is a Lyndon word}\}$,
i.e., $x[i..i + \lambda[i]]$ is the longest Lyndon prefix of suffix x_i
- (ii) $\forall i \in (1, n) : \text{prev}[i] = \max\{j \mid j \in [1, i) \wedge x_j \prec x_i\}$, $\text{prev}[1] = 0$, $\text{prev}[n] = 1$,
i.e., $x_{\text{prev}[i]}$ is the nearest suffix starting left of i that is lex. smaller than x_i
- (iii) $\forall i \in (1, n) : \text{next}[i] = \min\{j \mid j \in (i, n] \wedge x_j \prec x_i\}$, $\text{next}[1] = n + 1$, $\text{next}[n] = n + 1$,
i.e., $x_{\text{next}[i]}$ is the nearest suffix starting right of i that is lex. smaller than x_i

Figure 1a shows an example of these arrays. In drawings, we use a directed edge from position i to position j of a string to indicate that either $\text{prev}[i] = j$ (whenever the edge is directed from right to left) or $\text{next}[i] = j$ (whenever the edge is directed from left to right). We refer to these edges as PSS and NSS edges. It is no coincidence that in the example it holds $\text{next}[i] = i + \lambda[i]$ for all i . In fact, this is a fundamental combinatorial property of the Lyndon array, which was first (indirectly in a different form) shown by Hohlweg and Reutenauer [12]. Subsequently, Franek et al. [9, Lemma 15] and Franek and Liut [11, Lemma 1]² proved the property in the form stated below.

► **Lemma 3** ([12, 9, 11]). *Let $x = \#x(1..n)\$$ be a string with Lyndon array λ and next smaller suffix array next , then it holds $\forall i \in [1, n] : \text{next}[i] = i + \lambda[i]$.*

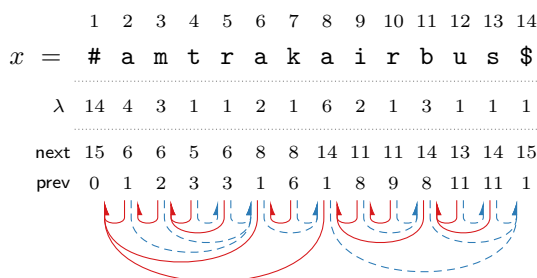
Another property shown by Bille et al. [5] relates prev and Lyndon words:

► **Lemma 4** ([5, Lemma 4]). *Let $x = \#x(1..n)\$$ be a string with previous smaller suffix array prev . For every $i \in [2, n]$, the string $x[\text{prev}[i]..i]$ is a Lyndon word.*

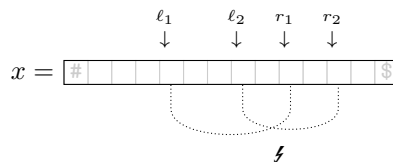
¹ see, e.g., lines 4–5 in file <https://github.com/jonas-ellert/simple-lyndon/blob/main/lyndon.hpp>

² Lemma 1 (b) in [11] should state “ $x[i..j]$ is proto-Lyndon” rather than “ $x[i..n]$ is proto-Lyndon”

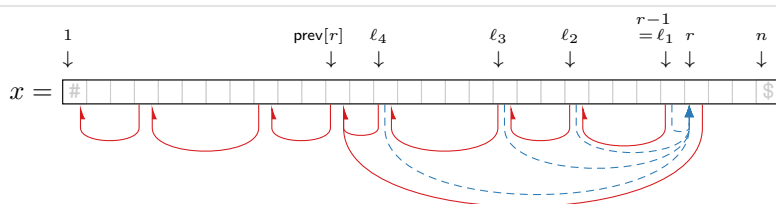
3 Key Properties of Nearest Smaller Suffixes



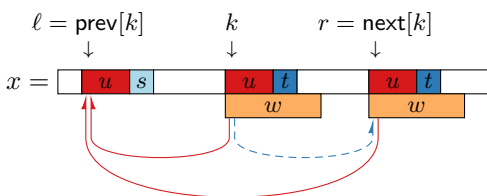
(a) Example of arrays λ , next and prev.



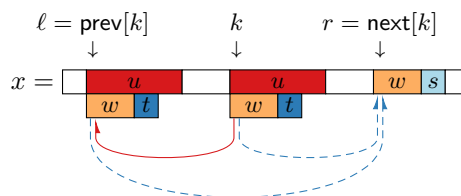
(b) Drawing for Lemma 5.



(c) Drawing for Lemma 6.



(d) Drawing for Lemma 7(ii).



(e) Drawing for Lemma 7(iii).

Figure 1 Examples and technical drawings for Sections 2 and 3. PSS edges are solid and red. NSS edges are dashed and blue. Dotted black edges are either NSS or PSS edges.

In this section, we show combinatorial properties of prev and next that are essential for all algorithms presented in this paper. Some of the properties have been shown (in a similar form) in [5, 1]. Proving Lemmas 5–7 is helpful for truly understanding the mechanisms at play. The reader is encouraged to do so on their own, with the help of the provided supplementary drawings. The full proofs are given in Section 7.

Naming of variables. Throughout the remainder of the paper, we use the variables ℓ and r to denote positions of the string. The intended meaning of these variables is *left* and *right*, i.e., whenever we use ℓ and r it holds $\ell < r$.

PSS and NSS edges are intersection-free. The first property that we show is that, when drawn underneath the text as in the previous example, none of the PSS and NSS edges intersect. This is formally expressed by the following lemma (see also Figure 1b).

► **Lemma 5.** *Let $x = \#x(1..n)\$$ be a string with previous and next smaller suffix array prev and next . Let $\ell_1, \ell_2, r_1, r_2 \in [1, n]$ be indices with either $\text{next}[\ell_1] = r_1$ or $\text{prev}[r_1] = \ell_1$, and also either $\text{next}[\ell_2] = r_2$ or $\text{prev}[r_2] = \ell_2$. Then it does not hold $\ell_1 < \ell_2 < r_1 < r_2$.*

Chains of previous smaller suffixes. For arbitrary index $r \in [1, n]$ and integer $e \geq 0$, we recursively define $\text{prev}^0[r] = r$ and $\text{prev}^{e+1}[r] = \text{prev}^e[\text{prev}[r]]$. We write $\ell = \text{prev}^*[r]$ to denote that there is some integer $e \geq 0$ with $\ell = \text{prev}^e[r]$. Note that generally $1 = \text{prev}^*[r]$ (due to the sentinel $x[1] = \#$) and $r = \text{prev}^*[r]$. In drawings, $\ell = \text{prev}^e[r]$ means that there is a chain of PSS edges from r to ℓ . The following lemma states useful properties of PSS chains, which we will later use to compute the arrays `prev` and `next`. The lemma is visualized in Figure 1c.

► **Lemma 6.** *Let $x = \#x(1..n)\$$ be a string with previous and next smaller suffix arrays `prev` and `next`, and let $\ell, r \in [1, n]$ be arbitrary indices.*

- (i) *It holds $\text{prev}[r] = \text{prev}^*[r - 1]$.*
- (ii) *It holds $\text{next}[\ell] = r$ if and only if $\ell = \text{prev}^*[r - 1]$ and $\ell > \text{prev}[r]$.*

Finally, for some indices that are related via `next` and `prev`, we can deduce LCEs. The following lemma is visualized in Figures 1d and 1e.

► **Lemma 7.** *Let $x = \#x(1..n)\$$ be a string with previous and next smaller suffix arrays `prev` and `next`. Let $k \in (1, n)$ be an arbitrary index, and let $\ell = \text{prev}[k]$ and $r = \text{next}[k]$.*

- (i) *If $\text{LCE}(\ell, k) = \text{LCE}(k, r)$, then $\text{LCE}(\ell, r) \geq \text{LCE}(k, r)$ and either $\text{prev}[r] = \ell$ or $\text{next}[\ell] = r$.*
- (ii) *If $\text{LCE}(\ell, k) < \text{LCE}(k, r)$, then $\text{LCE}(\ell, r) = \text{LCE}(\ell, k)$ and $\text{prev}[r] = \ell$.*
- (iii) *If $\text{LCE}(\ell, k) > \text{LCE}(k, r)$, then $\text{LCE}(\ell, r) = \text{LCE}(k, r)$ and $\text{next}[\ell] = r$.*

4 Algorithms to Compute the Lyndon Array

Due to Lemma 3, instead of designing algorithms that compute the Lyndon array λ , we can design algorithms that compute the next smaller suffix array `next`. This has been done, e.g., by Bille et al. [5] and Crochemore et al. [1], and is also the general approach used in this paper. All of the presented algorithms are based on a simple folklore algorithm for nearest smaller values, where instead of comparing values we lexicographically compare suffixes (Algorithm 1(a)). We obtain three different versions of this algorithm depending on how the lexicographical comparisons are implemented: Algorithm 1(b) uses the inverse suffix array and is only shown because it is a standard solution for computing the Lyndon array. Algorithm 1(c) implements lexicographical comparisons with naively computed LCEs. Algorithm 1(d) refines the LCE computation such that it is more time efficient. In the remainder of this section, we explain each version of the algorithm in detail.

Algorithms 1(c) and 1(d) require super-linear time, but they can be seen as incremental stepping stones towards the final solution. In Section 5, we modify Algorithm 1(d) such that it runs in $\mathcal{O}(n)$ time.

Algorithm 1: The general approach. Due to the sentinels, we can directly assign `prev[1]`, `next[1]`, and `next[n]` (lines 1–2). We compute the arrays `next` and `prev` in $n - 1$ iterations of a simple for-loop (line 4). The goal of iteration r is to compute `prev[r]`, while also identifying all indices ℓ with `next[ℓ] = r`. A simple strategy for this is dictated by Lemma 6, which states that all the relevant indices lie on the chain of PSS edges that starts at position $r - 1$. We thus inspect the positions $\ell = \text{prev}^*[r - 1]$ one at a time, starting with $\ell = r - 1$ (line 5). As long as $x_\ell \succ x_r$, we assign `next[ℓ] ← r`, and then continue with the next index $\ell \leftarrow \text{prev}[\ell]$ on the chain of PSS edges (lines 6–8). As soon as $x_\ell \prec x_r$, we break out of the inner loop and finish the current iteration of the outer loop by assigning `prev[r] ← ℓ` (line 9). (The sentinel $x[1] = \#$ ensures that $1 = \text{prev}^*[r]$ and $x_1 \prec x_r$; thus we are guaranteed to reach some ℓ with $x_\ell \prec x_r$ eventually.) The correctness of the algorithm follows directly from Lemma 6. An example of an outer loop iteration is provided in Figure 2 (the arrays `lexrank`, `plice`, and `nlice` will be relevant later and can be ignored for now).

■ **Algorithm 1** Various algorithms for computing nearest smaller suffixes.

Require: string $x = x[1..n] = \#x(i..n)\$$

Ensure: previous and next smaller suffix arrays `prev` and `next`

- 1: `prev[1..n]` \leftarrow new array with `prev[1] = 0`
- 2: `next[1..n]` \leftarrow new array with `next[1] = next[n] = n + 1`

(a) Folklore

```

3: —
4: for r = 2 to n do
5:   ℓ ← r - 1
6:   while xℓ > xr do
7:     next[ℓ] ← r
8:     ℓ ← prev[ℓ]
9:   prev[r] ← ℓ

```

(b) ISA-NSV

```

3: lexrank ← inverse suffix array of x
4: for r = 2 to n do
5:   ℓ ← r - 1
6:   while lexrank[ℓ] > lexrank[r] do
7:     next[ℓ] ← r
8:     ℓ ← prev[ℓ]
9:   prev[r] ← ℓ

```

(c) Naive LCE-NSS

```

3: plce[1..n] ← array filled with 0
4: nlce[1..n] ← array filled with 0

5: for r = 2 to n do
6:   ℓ ← r - 1
7:   m ← LCE-SCAN(ℓ, r)
8:   while x[ℓ + m] > x[r + m] do
9:     next[ℓ], nlce[ℓ] ← r, m
10:    m ← LCE-SCAN(prev[ℓ], r)
11:    —
12:    —
13:    —
14:   ℓ ← prev[ℓ]
15:   prev[r], plce[r] ← ℓ, m

```

(d) Improved LCE-NSS

```

3: plce[1..n] ← array filled with 0
4: nlce[1..n] ← array filled with 0

5: for r = 2 to n do
6:   ℓ ← r - 1
7:   m ← LCE-SCAN(ℓ, r)
8:   while x[ℓ + m] > x[r + m] do
9:     next[ℓ], nlce[ℓ] ← r, m
10:    if m = plce[ℓ] then
11:      m ← LCE-EXTEND(prev[ℓ], r, m)
12:    else if m > plce[ℓ] then
13:      m ← plce[ℓ]
14:   ℓ ← prev[ℓ]
15:   prev[r], plce[r] ← ℓ, m

```

(e) LCE Functions for (c) and (d)

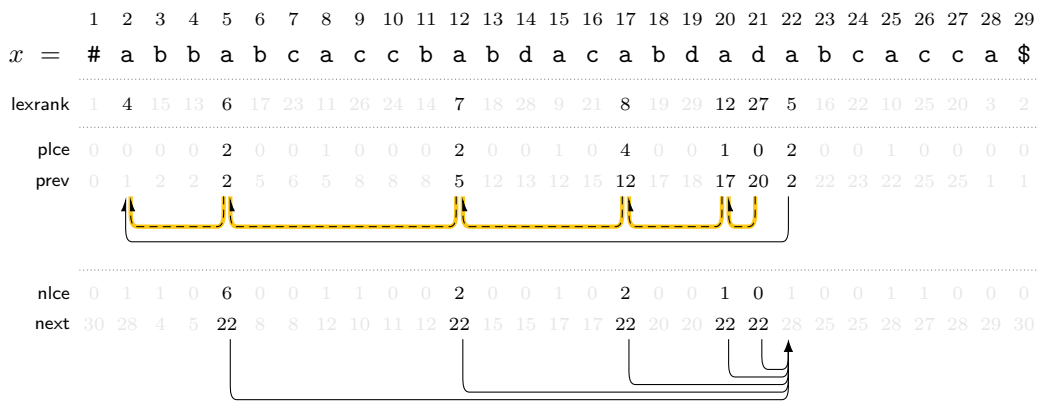
```

function LCE-EXTEND(ℓ, r, m)
  while x[ℓ + m] = x[r + m] do
    m ← m + 1
  return m

function LCE-SCAN(ℓ, r)
  return LCE-EXTEND(ℓ, r, 0)

```

to achieve $\mathcal{O}(n)$ time, substitute:
 line 7: $m \leftarrow \text{SMART-LCE}(\ell, r, 0)$
 line 11: $m \leftarrow \text{SMART-LCE}(\text{prev}[\ell], r, m)$
 (see Algorithm 2)



■ **Figure 2** For any of the Algorithms 1(a)–(d), we perform six suffix comparisons in outer loop iteration $r = 22$. We evaluate $x_\ell \succ x_r$ for each of the values $\ell = 21, 20, 17, 12, 5, 2$ (precisely in this order). For the first five values $\ell = 21, 20, 17, 12, 5$, we enter the body of the inner loop and thus assign $\text{next}[\ell] \leftarrow 22$. We break out of the inner loop by discovering that $x_2 \prec x_{22}$, after which we assign $\text{prev}[22] \leftarrow 2$. By design of the algorithm, and following Lemma 6, the values assumed by ℓ form a consecutive chain of PSS edges starting at position $r - 1 = 21$ (dashed edges).

Whenever we perform the lexicographical comparison of suffixes in line 6, we correctly assign either $\text{next}[\ell] \leftarrow r$ or $\text{prev}[r] \leftarrow \ell$ immediately afterwards. Since each entry of prev and next gets assigned exactly once, we perform exactly $2n - 3$ suffix comparisons; we enter the body of the inner loop exactly $n - 2$ times (once per entry of next , but not for $\text{next}[1] = \text{next}[n] = n + 1$), and break out of the inner loop exactly $n - 1$ times (once per outer loop iteration, or equivalently once per entry of prev , but not for $\text{prev}[1] = 0$). It follows that the algorithm takes $\mathcal{O}(n)$ time, plus the time needed to perform the suffix comparisons. Next, we discuss three possible implementations of these comparisons.

Algorithm 1(b): Using the inverse suffix array. The inverse suffix array $\text{lexrank}[1..n]$ of $x[1..n]$ is the unique permutation of $[1, n]$ that satisfies $\forall \ell, r \in [1, n] : x_\ell \prec x_r \iff \text{lexrank}[\ell] < \text{lexrank}[r]$ (an example is provided in Figure 2). Algorithm 1(b) precomputes the inverse suffix array, and then uses it to perform the lexicographical suffix comparisons. This idea was first proposed by Franek et al. [9, Algorithm NSVISA]. As discussed in Section 1, the precomputation takes $\mathcal{O}(n \lg n)$ time for general ordered alphabets, or $\mathcal{O}(n)$ time for linearly-sortable alphabets. The remainder of the algorithm takes $\mathcal{O}(n)$ time because we perform $\mathcal{O}(n)$ suffix comparisons, each of which takes constant time when using lexrank .

Algorithm 1(c): Using LCEs with simple scanning. If $\ell \neq r$ and $x[n] = \$$, then it holds $x_\ell \prec x_r \iff x[\ell + \text{LCE}(\ell, r)] < x[r + \text{LCE}(\ell, r)]$ (both of the conditions are satisfied for the comparisons performed by our algorithms). The LCE can be computed by simple scanning, as shown in Algorithm 1(e). Due to the sentinel $x[n] = \$$, no suffix is prefix of another suffix, and we always find a mismatching symbol eventually. Algorithm 1(c) implements the lexicographical suffix comparisons with LCEs (lines 7, 8, and 10). Additionally, it stores the computed LCEs in two arrays nlice and plice (lines 3–4, 9, and 15), where after termination it holds $\text{plice}[i] = \text{LCE}(\text{prev}[i], i)$ and $\text{nlice}[i] = \text{LCE}(i, \text{next}[i])$ for all $i \in (1, n)$. These arrays are of independent interest. For example, nlice is useful when computing maximal repetitions [8].

Computing some $\text{LCE}(\ell, r)$ by scanning takes $\text{LCE}(\ell, r) + 1$ symbol comparisons: $\text{LCE}(\ell, r)$ comparisons with outcome “equal”, and one comparison with outcome “not equal”. For the example iteration in Figure 2, we compute $\text{LCE}(21, 22) = 0$, $\text{LCE}(20, 22) = 1$, $\text{LCE}(17, 22) = 2$,

$\text{LCE}(12, 22) = 2$, $\text{LCE}(5, 22) = 6$ and $\text{LCE}(17, 22) = 2$, and thus we perform 19 symbol comparisons. In the worst case, a single LCE scan takes $\mathcal{O}(n)$ time, and thus Algorithm 1(c) takes $\mathcal{O}(n^2)$ time (the bound is tight, e.g., for the string $x = \#a^{n-2}\$$). If the string x is drawn uniformly at random from the set of length- n strings over Σ , where $|\Sigma| > 1$, then the expected running time of Algorithm 1(c) is $\mathcal{O}(n)$ (see [1, Theorem 7]).

Algorithm 1(d): Using LCEs with improved scanning. In a single outer loop iteration r of Algorithm 1(c), we may compute $\text{LCE}(\ell, r)$ for multiple different values of ℓ . So far, we always scanned each new LCE entirely from scratch (line 10). For many of the LCEs, we can avoid (a part of) the scan by utilizing Lemma 7. This is done in Algorithm 1(d), which is identical to Algorithm 1(c), except for the highlighted computation of the LCE in lines 10–13. At the point in time at which we reach line 10, let $k' = \ell$, $\ell' = \text{prev}[\ell]$, and $r' = r$. Note that $\ell' = \text{prev}[k']$ and $r' = \text{next}[k']$, and thus we can use ℓ' , k' , and r' to invoke Lemma 7. We already computed $\text{LCE}(\ell', k') = \text{plce}[k']$ (due to the iteration order of the algorithm) and $\text{LCE}(k', r') = \text{nlce}[k'] = m$ (this is the most recently computed LCE). Now we compute $\text{LCE}(\ell', r')$ according to the cases of Lemma 7:

- If $\text{LCE}(\ell', k') = \text{LCE}(k', r')$ then Lemma 7(i) implies $\text{LCE}(\ell', r') \geq \text{LCE}(k', r')$. We compute $\text{LCE}(\ell', r')$ by scanning, but we skip $m = \text{LCE}(k', r')$ symbol comparisons (lines 10–11).
- If $\text{LCE}(\ell', k') < \text{LCE}(k', r')$ then Lemma 7(ii) implies $\text{LCE}(\ell', r') = \text{LCE}(\ell', k')$. Since $\text{plce}[k'] = \text{LCE}(\ell', k')$, we can simply assign $m \leftarrow \text{plce}[k']$ (lines 12–13). Note that Lemma 7(ii) also implies $\text{prev}[r'] = \ell'$, which means that we will immediately break out of the inner loop and finish the current iteration of the outer loop.
- If $\text{LCE}(\ell', k') > \text{LCE}(k', r')$ then Lemma 7(iii) implies $\text{LCE}(\ell', r') = \text{LCE}(k', r')$. It already holds $m = \text{LCE}(k', r')$, and thus there is no need to do anything.

For the example iteration in Figure 2, we entirely skip the computation of $\text{LCE}(12, 22)$ due to Lemma 7(iii), as well as the computation of $\text{LCE}(2, 22)$ due to Lemma 7(ii). Additionally, we skip two symbol comparisons when computing $\text{LCE}(5, 22)$, and one symbol comparison when computing $\text{LCE}(17, 22)$. The number of symbol comparisons for iteration 22 is 10 (significantly less than the 19 comparisons needed by Algorithm 1(c)). However, Algorithm 1(d) still takes $\mathcal{O}(n^2)$ time in the worst case. In the next section, we slightly modify the algorithm such that it achieves linear time.

A note on the space complexity. Algorithms 1(c) and 1(d) require $4n \lceil \log_2 n \rceil$ bits to store the arrays `next`, `prev`, `nlce` and `plce`. For a small practical improvement, it is possible to remove the array `prev`. This is because the only access to `prev`[ℓ] occurs at the same time at which we assign `next`[ℓ] (see lines 9 and 14). Thus, we only need to maintain access to the values `prev`[ℓ] for positions with uninitialized `next`[ℓ], which means that we can use a single array for storing both PSS and NSS information. The total working space (without the input string) then becomes $3n \lceil \log_2 n \rceil + \mathcal{O}(\lg n)$ bits.

5 Achieving Linear Time

In order to achieve linear time, we use the function SMART-LCE (Algorithm 2) to more efficiently compute LCEs. A call to `SMART-LCE`(ℓ, r, m) means that we want to compute $\text{LCE}(\ell, r)$, and we have already established $\text{LCE}(\ell, r) \geq m$. We modify Algorithm 1(d) by replacing line 7 with $m \leftarrow \text{SMART-LCE}(\ell, r, 0)$, and line 11 with $m \leftarrow \text{SMART-LCE}(\text{prev}[\ell], r, m)$ (and leave everything else unchanged). In the remainder of the section, we show that SMART-LCE works correctly, and that the total time spent for all invocations of SMART-LCE is $\mathcal{O}(n)$. Then, it directly follows that the modified version of Algorithm 1(d) takes $\mathcal{O}(n)$ time. Note that Algorithm 2 is tailored to (and thus only works as a part of) Algorithm 1(d).

■ **Algorithm 2** Efficient LCE computation (only works in conjunction with Algorithm 1(d)).

Require: string $x = x[1..n] = \#x(i..n)\$$ with $x[\ell..r + m] = x[r..r + m]$.

Ensure: longest common extension $\text{LCE}(\ell, r)$

```

1: global variable  $c \leftarrow 0$ 
2: global variable  $d \leftarrow 0$ 
3: function SMART-LCE( $\ell, r, m$ )
4:   if  $r + m < c$  then
5:     if  $\text{next}[\ell - d] = r - d$  then  $m \leftarrow \text{nlce}[\ell - d]$ 
6:       else  $m \leftarrow \text{plce}[r - d]$ 
7:     if  $r + m < c$  then return  $m$ 
8:      $m \leftarrow c - r$ 
9:   while  $x[\ell + m] = x[r + m]$  do
10:     $m \leftarrow m + 1$ 
11:    $c, d \leftarrow r + m, r - \ell$ 
12:   return  $m$ 

```

In the following description, whenever we use the variables ℓ , r , and m , we mean the arguments of the function SMART-LCE (rather than the identically named variables from Algorithm 1(d)). Now we explain how the new LCE function works. Generally speaking, it computes LCEs with two different methods: naive scanning (as done before), and deduction from previously computed LCEs. Sometimes, a combination of both is necessary. Both methods rely on a global variable c (persistent between the function calls) that stores at all times the rightmost position of the string that we have already inspected (line 2).

Scanning LCEs. We start by explaining the simpler method of naive scanning. If at the beginning of the function call it holds $r + m \geq c$ (line 4), then we simply scan the remainder of the LCE (lines 9–10; identical to what we did in LCE-EXTEND). Let m' be the initial value of m before the scan, and let $m'' = \text{LCE}(\ell, r)$ be the final value of m after performing the scan. After the scan, the rightmost inspected position is $r + m''$, and we update c accordingly (line 11; the variable d is not relevant for now). Since we only perform the scan if $r + m' \geq c$, the assignment $c \leftarrow r + m''$ increases c by at least $m'' - m'$. Note that $m'' - m'$ is also exactly the number of times we execute line 10. Since c never exceeds n , we execute line 10 no more than n times during all the calls to SMART-LCE that initially satisfy $r + m \geq c$. It follows that, for all of these calls together, we spend at most $\mathcal{O}(n)$ time.

Deducing LCEs. If at the beginning of the function call it holds $r + m < c$, then we try to deduce $\text{LCE}(\ell, r)$ from previously computed LCEs (lines 4–8). Let r_c be the rightmost position for which we already computed some $\text{LCE}(\ell_c, r_c)$ with $r_c + \text{LCE}(\ell_c, r_c) = c$ (such a position must exist because otherwise we would not have inspected $x[c]$ yet). The global variable d contains at all times the distance $r_c - \ell_c$ (line 2; we update d together with c , see line 11). Let $\ell_* = \ell - d$ and $r_* = r - d$. The example in Figure 3 helps with understanding the notation. Later, we will show that (as suggested by the examples)

- (i) it holds $r_c \leq \ell < r < c$, and thus $\ell_c \leq \ell_* < r_* < \ell_c + \text{LCE}(\ell_c, r_c)$, and
- (ii) either $\text{prev}[r_*] = \ell_*$ (and thus $\text{plce}[r_*] = \text{LCE}(\ell_*, r_*)$)
or $\text{next}[\ell_*] = r_*$ (and thus $\text{nlce}[\ell_*] = \text{LCE}(\ell_*, r_*)$).

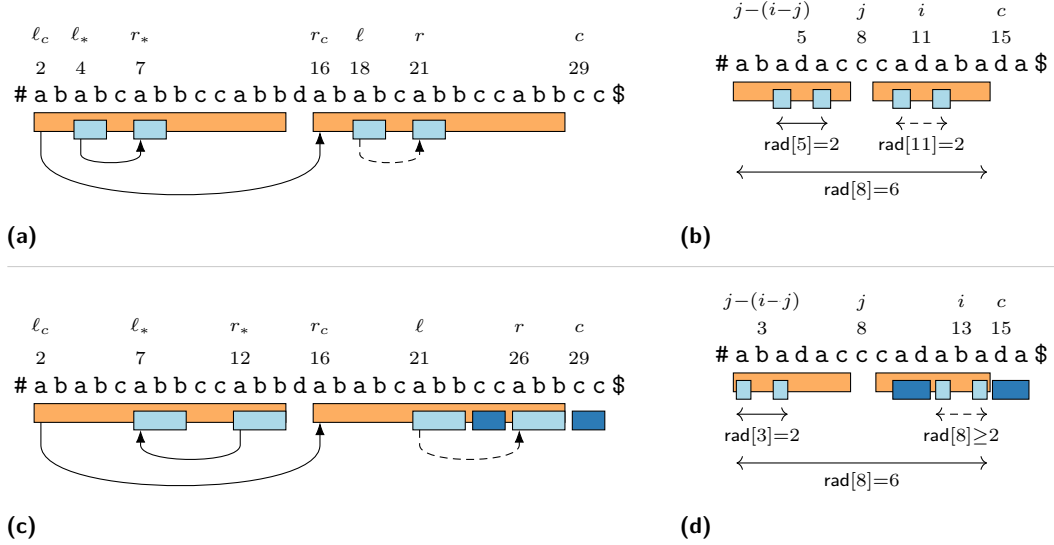


Figure 3 Deducing LCEs with Algorithm 2 in (a) and (c), and deducing longest palindromes with Manacher’s algorithm in (b) and (d). Boxes of equal color indicate equal substrings. In (b) and (d), boxes of equal color sometimes indicate substrings that are the reverse of each other.

When deducing LCEs, we first use (ii) to obtain $\text{LCE}(\ell_*, r_*)$ (lines 5–6). Note that, by the definition of ℓ_* and r_* , the relative positions of ℓ_* and r_* within $x[\ell_c.. \ell_c + \text{LCE}(\ell_c, r_c)]$ are the same as the relative positions of ℓ and r within $x[r_c.. r_c + \text{LCE}(\ell_c, r_c)]$ (and the positions are indeed within these intervals due to (i)). If $r + \text{LCE}(\ell_*, r_*) < c$ then

$$x[r..r + \text{LCE}(\ell_*, r_*)] = x[r_*..r_* + \text{LCE}(\ell_*, r_*)] \text{ and}$$

$$x[\ell..\ell + \text{LCE}(\ell_*, r_*)] = x[\ell_*..\ell_* + \text{LCE}(\ell_*, r_*)],$$

where both equalities follow from $x[\ell_c.. \ell_c + \text{LCE}(\ell_c, r_c)] = x[r_c.. r_c + \text{LCE}(\ell_c, r_c)]$. This implies $\text{LCE}(\ell, r) = \text{LCE}(\ell_*, r_*)$, and we return $\text{LCE}(\ell_*, r_*)$ in constant time (line 7). Since it holds $r + \text{LCE}(\ell, r) < c$, there is no need to update c and d . In Figure 3a, we have $21 + \text{LCE}(4, 7) = 23 < 29 = c$, and thus $\text{LCE}(18, 21) = \text{LCE}(4, 7) = 2$.

If, however, $r + \text{LCE}(\ell_*, r_*) \geq c$ then we cannot immediately deduce the exact value of $\text{LCE}(\ell, r)$ (as is the case in Figure 3c). We can still obtain some useful information because of

$$x[r..r + c - r] = x[r_*..r_* + c - r] = x[\ell_*..\ell_* + c - r] = x[\ell..\ell + c - r],$$

where the first and the third equality follow from $x[\ell_c.. \ell_c + \text{LCE}(\ell_c, r_c)] = x[r_c.. r_c + \text{LCE}(\ell_c, r_c)]$, and the second equality follows from $r + \text{LCE}(\ell_*, r_*) \geq c$, which is equal to $\text{LCE}(\ell_*, r_*) \geq c - r$. The equation implies $\text{LCE}(\ell, r) \geq c - r$, and we update m accordingly (line 8). In Figure 3c, we have $26 + \text{LCE}(7, 12) = 29 = c$, and thus $\text{LCE}(21, 26) \geq 29 - 26 = 3$.

We compute the remaining part of $\text{LCE}(\ell, r)$ by scanning (lines 9–10), and then update c and d (line 11). Since we assign $m \leftarrow (c - r)$ immediately before starting the scan, we can use the same argument as in the previous paragraph about scanning LCEs: For every symbol comparison of the scan (except for the last one), we will increase c by one. Therefore, the total number of symbol comparisons for all calls of SMART-LCE is $\mathcal{O}(n)$. In Figure 3c, the scan extends the LCE by two more positions, and we obtain $\text{LCE}(21, 26) = 5$. We then have to update $c \leftarrow 26 + 5 = 31$ and $d \leftarrow 26 - 21 = 5$.

The correctness of the algorithm follows from its description and the properties (i) and (ii), which we will show in the next paragraphs.

Showing Property (i). The property states that, if we call $\text{SMART-LCE}(\ell, r, m)$ with $r + m < c$, then $r_c \leq \ell < r < c$. Since trivially $\ell < r \leq r + m < c$, we only have to show $r_c \leq \ell$. The property is readily proven for the call $\text{SMART-LCE}(\ell, r, 0)$ in line 7 of Algorithm 1(d). It holds $\ell = r - 1$, and this is the first LCE that we compute between r and any smaller index. Since we already computed $\text{LCE}(\ell_c, r_c)$, it holds $r > r_c$ and $\ell = r - 1 \geq r_c$.

Now we consider the call $\text{SMART-LCE}(\ell, r, m)$ in line 11. As seen in the description of Algorithm 1(d), for this call it holds $m = \text{LCE}(\ell, k) = \text{LCE}(k, r)$, where $k \in (\ell, r)$ with $\text{prev}[k] = \ell$ and $\text{next}[k] = r$. For every $h \in (\ell, r)$, the definition of prev and next implies that $x_h \succeq x_k \succ x_r$. This also means that $m = \text{LCE}(k, r) \geq \text{LCE}(h, r)$. If $r = r_c$ then, because we already computed $\text{LCE}(\ell_c, r)$, and due to the iteration order of the algorithm, it holds $\ell < \ell_c$. Then, however, $\ell_c \in (\ell, r)$ and thus $m = \text{LCE}(k, r) \geq \text{LCE}(\ell_c, r) = c - r$, which contradicts $r + m < c$. We have shown that $r > r_c$, which also implies $r_* > \ell_c$. If $\ell < r_*$ then $r_* \in (\ell, r)$ and thus $m \geq \text{LCE}(r_*, r) = c - r$, which contradicts $r + m < c$. It follows that $\ell \geq r_* > \ell_c$. Finally, if $\ell \in (\ell_c, r_c)$ then $\ell_c < \ell < r_c < r$, which contradicts Lemma 5. The only remaining possibility is $\ell \geq r_c$, which is what we wanted to show.

Showing Property (ii). The property states that either $\text{prev}[r_*] = \ell_*$, or $\text{next}[\ell_*] = r_*$. By the definition of ℓ_* and r_* , and due to (i) and $x[\ell_c.. \ell_c + \text{LCE}(\ell_c, r_c)] = x[r_c..r_c + \text{LCE}(\ell_c, r_c)]$, it holds $x[\ell..r] = x[\ell_*..r_*]$. Since we want to compute $\text{LCE}(\ell, r)$, it holds either $\text{next}[\ell] = r$ or $\text{prev}[r] = \ell$. Therefore, either Lemma 3 or Lemma 4 implies that $x[\ell..r] = x[\ell_*..r_*]$ is a Lyndon word. Due to Lemma 3, we know that $\text{next}[\ell_*] \geq r_*$. If $\text{next}[\ell_*] = r_*$ or $\text{prev}[r_*] = \ell_*$, then there is nothing left to show. Thus, assume that $\text{next}[\ell_*] > r_*$ and $\text{prev}[r_*] > \ell_*$ (it cannot be that $\text{prev}[r_*] < \ell_*$ because then $\text{prev}[r_*] < \ell_* < r_* < \text{next}[\ell_*]$ contradicts Lemma 5). Let $p_r = r - (r_* - \text{prev}[r_*])$. Due to Lemma 4, the substring $x[\text{prev}[r_*]..r_*] = x[p_r..r]$ is a Lyndon word, and Lemma 3 implies $\text{next}[p_r] \geq r$. Since $p_r \in (\ell, r)$ it holds $\text{next}[p_r] \leq r$ (otherwise we contradict Lemma 5), and the only possible option is $\text{next}[p_r] = r$.

We have shown that $\text{next}[p_r] = r$ and thus $x_{p_r} \succ x_r$. By the definition of prev , it also holds $x_{\text{prev}[r_*]} \prec x_{r_*}$. Since we chose r_c to be the rightmost index with $r_c + \text{LCE}(\ell_c, r_c) = c$, it holds $r + \text{LCE}(p_r, r) < c$ (otherwise we would have updated r_c already). Therefore, we have

$$\begin{aligned} x[\text{prev}[r_*].. \text{prev}[r_*] + \text{LCE}(p_r, r)] &= x[p_r..p_r + \text{LCE}(p_r, r)], \text{ and} \\ x[r_*..r_* + \text{LCE}(p_r, r)] &= x[r..r + \text{LCE}(p_r, r)]. \end{aligned}$$

This, however, means that $x_{\text{prev}[r_*]} \prec x_{r_*} \iff x_{p_r} \prec x_r$, which contradicts our previous observation that $x_{p_r} \succ x_r$ and $x_{\text{prev}[r_*]} \prec x_{r_*}$. It follows that the assumption $\text{next}[\ell_*] > r_*$ and $\text{prev}[r_*] > \ell_*$ was wrong, and it holds $\text{next}[\ell_*] = r_*$ or $\text{prev}[r_*] = \ell_*$.

6 Similarity to Manacher's Algorithm

In this section, we want to briefly highlight the similarity between the technique of Section 5 and Manacher's algorithm for computing maximal palindromes [14]. For simplicity, we only consider odd palindromes. An *odd palindrome of radius* $|w| + 1$ is a string of the form $ws\bar{w}$, where s is a symbol, w is some possibly empty string, and \bar{w} is the *reverse string* of w defined by $|\bar{w}| = |w|$ and $\forall i \in [1, |w|] : \bar{w}[i] = w[|w| - i + 1]$. For a string $x[1..n]$, the presented version of Manacher's algorithm computes an array $\text{rad}[1..n]$, where $\forall i \in [1, n] :$

$$\text{rad}[i] = \max\{m \mid m \in [1, \min(i, n - i + 1)] \text{ and } x(i - m..i + m) \text{ is an odd palindrome}\}.$$

If we compute the entries of rad in left-to-right order, then we can sometimes fully or partially

■ **Algorithm 3** Manacher’s algorithm for odd palindromes.

<p>Require: string $x = \#x(1..n)\\$.</p> <p>Ensure: array rad containing the radii of the longest odd palindromes</p> <p>1: $\text{rad}[1..n] \leftarrow$ new array initialized with $\text{rad}[1] = \text{rad}[n] = 1$</p> <p>2: global variable $c \leftarrow 0$</p> <p>3: global variable $j \leftarrow 0$</p> <p>4: for $i = 2$ to $n - 1$ do</p> <p>5: $\text{rad}[i] \leftarrow \text{SMART-PALINDROME}(i, 1)$</p>	<p>1: function $\text{SMART-PALINDROME}(i, m)$</p> <p>2: if $i + m < c$ then</p> <p>3: $m \leftarrow \text{rad}[j - (i - j)]$</p> <p>4: if $i + m < c$ then return m</p> <p>5: $m \leftarrow c - i$</p> <p>6: while $x[i - m] = x[i + m]$ do</p> <p>7: $m \leftarrow m + 1$</p> <p>8: $c, j \leftarrow i + m, i$</p> <p>9: return m</p>
--	---

deduce an entry, see Figures 3b and 3d. This is highly similar to our observations for LCEs in Figures 3a and 3c. A possible implementation of Manacher’s algorithm is provided in Algorithm 3. It computes rad from left to right, while keeping track of the rightmost inspected position of the string. Whenever possible, the function SMART-PALINDROME partially or fully deduces $\text{rad}[i]$. Note that the functions SMART-LCE and SMART-PALINDROME are structurally identical and use the same algorithmic ideas. Due to space constraints, we omit further details on how and why Algorithm 3 functions as intended, and why it takes $\mathcal{O}(n)$ time. (However, we invite the reader to produce a proof of correctness on their own. This is much easier for Algorithm 3 than for Algorithm 1(d) and Algorithm 2. Particularly, it requires no complicated technicalities like properties (i) and (ii) in Section 5.)

7 Proofs for Section 3

► **Lemma 5.** *Let $x = \#x(1..n)\$$ be a string with previous and next smaller suffix array prev and next . Let $\ell_1, \ell_2, r_1, r_2 \in [1, n]$ be indices with either $\text{next}[\ell_1] = r_1$ or $\text{prev}[r_1] = \ell_1$, and also either $\text{next}[\ell_2] = r_2$ or $\text{prev}[r_2] = \ell_2$. Then it does not hold $\ell_1 < \ell_2 < r_1 < r_2$.*

Proof. Assume $\ell_1 < \ell_2 < r_1 < r_2$. Since $\ell_2 \in (\ell_1, r_1)$ and either $\text{next}[\ell_1] = r_1$ or $\text{prev}[r_1] = \ell_1$, Definition 2 implies $x_{\ell_2} \succ x_{r_1}$. However, it also holds $r_1 \in (\ell_2, r_2)$ and either $\text{next}[\ell_2] = r_2$ or $\text{prev}[r_2] = \ell_2$. Thus, Definition 2 also implies $x_{\ell_2} \prec x_{r_1}$, which is a contradiction. ◀

► **Lemma 6.** *Let $x = \#x(1..n)\$$ be a string with previous and next smaller suffix arrays prev and next , and let $\ell, r \in [1, n]$ be arbitrary indices.*

(i) *It holds $\text{prev}[r] = \text{prev}^*[r - 1]$.*

(ii) *It holds $\text{next}[\ell] = r$ if and only if $\ell = \text{prev}^*[r - 1]$ and $\ell > \text{prev}[r]$.*

Proof. For (i), assume that $\text{prev}[r] \neq \text{prev}^*[r - 1]$. Then there must be some $r' = \text{prev}^*[r - 1]$ with $\text{prev}[r] \in (\text{prev}[r'], r')$. However, $\text{prev}[r'] < \text{prev}[r] < r' < r$ contradicts Lemma 5.

For (ii), we show both directions separately. Assume that $\text{next}[\ell] = r$ then $\forall k \in [\ell, r) : x_k \succ x_r$, which means $\text{prev}[r] \notin [\ell, r)$ and thus $\text{prev}[r] < \ell$. Assume that $\ell \neq \text{prev}^*[r - 1]$, then there must be some $r' = \text{prev}^*[r - 1]$ with $\ell \in (\text{prev}[r'], r')$. However, then $\text{prev}[r'] < \ell < r' < r$ and Lemma 5 contradict the assumption that $\text{next}[\ell] = r$. Thus $\ell = \text{prev}^*[r - 1]$.

For the counter direction, assume that $\text{prev}[r] < \ell$ and $\ell = \text{prev}^*[r - 1]$. By the definition of prev , and due to $\ell \in (\text{prev}[r], r)$, it holds $x_\ell \succ x_r$. It is easy to see that $\ell = \text{prev}^*[r - 1]$ implies $\forall k \in (\ell, r) : x_k \succ x_\ell$ (when following a chain of PSS edges, by definition of prev , the visited suffixes are lexicographically decreasing, and all suffixes skipped by a PSS edge are lexicographically larger). From $x_\ell \succ x_r$ and $\forall k \in (\ell, r) : x_k \succ x_\ell$ follows $\text{next}[\ell] = r$. ◀

► **Lemma 7.** Let $x = \#x(1..n)\$$ be a string with previous and next smaller suffix arrays prev and next . Let $k \in (1, n)$ be an arbitrary index, and let $\ell = \text{prev}[k]$ and $r = \text{next}[k]$.

- (i) If $LCE(\ell, k) = LCE(k, r)$, then $LCE(\ell, r) \geq LCE(k, r)$ and either $\text{prev}[r] = \ell$ or $\text{next}[\ell] = r$.
- (ii) If $LCE(\ell, k) < LCE(k, r)$, then $LCE(\ell, r) = LCE(\ell, k)$ and $\text{prev}[r] = \ell$.
- (iii) If $LCE(\ell, k) > LCE(k, r)$, then $LCE(\ell, r) = LCE(k, r)$ and $\text{next}[\ell] = r$.

Proof. We start with (i). Definition 2 implies $\forall i \in (\ell, k) \cup (k, r) : x_k \prec x_i$. Since $\ell = \text{prev}[k]$ we have $x_\ell \prec x_k$ and thus $\forall i \in (\ell, r) : x_\ell \prec x_i$. Analogously, due to $r = \text{next}[k]$ we have $\forall i \in (\ell, r) : x_r \prec x_i$. Thus, if $x_\ell \prec x_r$ then $\text{prev}[r] = \ell$, and if $x_\ell \succ x_r$ then $\text{next}[\ell] = r$.

For showing (ii), let $u = x[\ell.. \ell + LCE(\ell, k)]$, $s = x[\ell + LCE(\ell, k)]$, and $t = x[k + LCE(\ell, k)]$. By the definition of LCEs, it holds $s \neq t$. Due to $\ell = \text{prev}[k]$ we have $us \cdot x_{\ell+|us|} = x_\ell \prec x_k = ut \cdot x_{k+|ut|}$, and therefore $s < t$. Because of $LCE(\ell, k) < LCE(k, r)$, suffix x_r has prefix ut . Thus, it holds $x_\ell = us \cdot x_{\ell+|us|} \prec ut \cdot x_{r+|ut|} = x_r$. Due to (i), this means $\text{next}[\ell] = r$. The proof of (iii) works analogously to the one for (ii). ◀

References

- 1 Golnaz Badkobeh, Maxime Crochemore, Jonas Ellert, and Cyril Nicaud. Back-to-front online Lyndon forest construction. In *Proceedings of the 33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022)*, pages 13:1–13:23, Prague, Czech Republic, June 2022. doi:10.4230/LIPIcs.CPM.2022.13.
- 2 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)*, pages 23:1–23:12, Tel Aviv, Israel, June 2016. doi:10.4230/LIPIcs.CPM.2016.23.
- 3 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The “runs” theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
- 4 Nico Bertram, Jonas Ellert, and Johannes Fischer. Lyndon words accelerate suffix sorting. In *Proceedings of the 29th Annual European Symposium on Algorithms (ESA 2021)*, pages 15:1–15:13, Lisbon, Portugal (Virtual Conference), September 2021. doi:10.4230/LIPIcs.ESA.2021.15.
- 5 Philip Bille, Jonas Ellert, Johannes Fischer, Inge Li Gørtz, Florian Kurpicz, J. Ian Munro, and Eva Rotenberg. Space efficient construction of Lyndon arrays in linear time. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, pages 14:1–14:18, Saarbrücken, Germany (Virtual Conference), July 2020. doi:10.4230/LIPIcs.ICALP.2020.14.
- 6 Maxime Crochemore, Thierry Lecroq, and Wojciech Rytter. *125 Problems in Text Algorithms*. Cambridge University Press, 2021. 334 pages.
- 7 Jean-Pierre Duval. Factorizing words over an ordered alphabet. *J. Algorithms*, 4(4):363–381, 1983. doi:10.1016/0196-6774(83)90017-2.
- 8 Jonas Ellert and Johannes Fischer. Linear time runs over general ordered alphabets. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, pages 63:1–63:16, Glasgow, Scotland (Virtual Conference), July 2021. doi:10.4230/LIPIcs.ICALP.2021.63.
- 9 Frantisek Franek, A. S. M. Sohiddul Islam, Mohammad Sohel Rahman, and William F. Smyth. Algorithms to compute the Lyndon array. In *Proceedings of the Prague Stringology Conference 2016*, pages 172–184, Prague, Czech Republic, August 2016. URL: <http://www.stringology.org/event/2016/p15.html>.
- 10 Frantisek Franek and Michael Liut. Algorithms to compute the Lyndon array revisited. In *Proceedings of the Prague Stringology Conference 2019*, pages 16–28, Prague, Czech Republic, 2019. URL: <http://www.stringology.org/event/2019/p03.html>.

- 11 Frantisek Franek and Michael Liut. Computing maximal Lyndon substrings of a string. *Algorithms*, 13(11), 2020. doi:10.3390/a13110294.
- 12 Christophe Hohlweg and Christophe Reutenauer. Lyndon words, permutations and trees. *Theor. Comput. Sci.*, 307(1):173–178, 2003. doi:10.1016/S0304-3975(03)00099-9.
- 13 Felipe A. Louza, Sabrina Mantaci, Giovanni Manzini, Marinella Sciortino, and Guilherme P. Telles. Inducing the Lyndon array. In *Proceedings of the 26th International Symposium on String Processing and Information Retrieval (SPIRE 2019)*, pages 138–151, Segovia, Spain, October 2019. doi:10.1007/978-3-030-32686-9_10.
- 14 Glenn Manacher. A new linear-time “on-line” algorithm for finding the smallest initial palindrome of a string. *Journal of the ACM*, 22(3):346–351, 1975. doi:10.1145/321892.321896.
- 15 Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. In *Proceedings of the First Annual Symposium on Discrete Algorithms (SODA 1990)*, pages 319–327, San Francisco, California, USA, January 1990. URL: <http://dl.acm.org/citation.cfm?id=320176.320218>.
- 16 Kazuya Tsuruta, Dominik Köppl, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Grammar-compressed self-index with Lyndon words. *CoRR*, abs/2004.05309, 2020. arXiv:2004.05309.

Learning-Augmented Query Policies for Minimum Spanning Tree with Uncertainty

Thomas Erlebach   

Department of Computer Science, Durham University, UK

Murilo Santos de Lima   

München, Germany

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

We study how to utilize (possibly erroneous) predictions in a model for computing under uncertainty in which an algorithm can query unknown data. Our aim is to minimize the number of queries needed to solve the minimum spanning tree problem, a fundamental combinatorial optimization problem that has been central also to the research area of explorable uncertainty. For all integral $\gamma \geq 2$, we present algorithms that are γ -robust and $(1 + \frac{1}{\gamma})$ -consistent, meaning that they use at most γOPT queries if the predictions are arbitrarily wrong and at most $(1 + \frac{1}{\gamma})\text{OPT}$ queries if the predictions are correct, where OPT is the optimal number of queries for the given instance. Moreover, we show that this trade-off is best possible. Furthermore, we argue that a suitably defined *hop distance* is a useful measure for the amount of prediction error and design algorithms with performance guarantees that degrade smoothly with the hop distance. We also show that the predictions are PAC-learnable in our model. Our results demonstrate that untrusted predictions can circumvent the known lower bound of 2, without any degradation of the worst-case ratio. To obtain our results, we provide new structural insights for the minimum spanning tree problem that might be useful in the context of query-based algorithms regardless of predictions. In particular, we generalize the concept of witness sets – the key to lower-bounding the optimum – by proposing novel global witness set structures and completely new ways of adaptively using those.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases explorable uncertainty, queries, untrusted predictions

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.49

Related Version *Full Version:* <https://arxiv.org/abs/2206.15201>

Funding *Thomas Erlebach:* Supported by EPSRC grant EP/S033483/2.

Murilo Santos de Lima: Work done while employed at University of Leicester, United Kingdom, and 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.

1 Introduction

We introduce learning-augmented algorithms to the area of optimization under explorable uncertainty and focus on the fundamental *minimum spanning tree (MST) problem under explorable uncertainty*. We are given a (multi)graph $G = (V, E)$ with unknown edge weights $w_e \in \mathbb{R}_+$, for $e \in E$. For each edge e , an uncertainty interval I_e is known that contains w_e . A *query* on edge e reveals the true value w_e . The task is to determine an MST, i.e., a tree



© Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlöter; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 49; pp. 49:1–49:18

Leibniz International Proceedings in Informatics



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

that connects all vertices of G , of minimum total weight w.r.t. the true values w_e . A query set is called *feasible* if it reveals sufficient information to identify an MST (not necessarily its exact weight). As queries are costly, the goal is to find a feasible query set of minimum size.

We study *adaptive strategies* that make queries sequentially and utilize the results of previous steps to decide upon the next query. As there exist input instances that are impossible to solve without querying all edges, we evaluate our algorithms in an *instance-dependent* manner: For each input, we compare the number of queries made by an algorithm with the best possible number of queries *for that input*, using competitive analysis. For a given problem instance, let OPT denote an arbitrary optimal query set (we later give a formal definition of OPT). An algorithm is ρ -*competitive* if it executes, for any problem instance, at most $\rho \cdot |\text{OPT}|$ queries. While MST under explorable uncertainty is not a classical online problem where the input is revealed passively over time, the query results are uncertain and, to a large degree, dictate whether decisions to query certain edges were good or not. For analyzing an algorithm, it is natural to assume that the query results are determined by an adversary. This gives the problem a clear online flavor and prohibits the existence of 1-competitive algorithms even if we have unlimited running time and space [24]. We note that competitive algorithms in general do not have any running time requirements, but all our algorithm run in polynomial time.

The MST problem is among the most widely studied problems in the research area of *explorable uncertainty* [35] and has been a cornerstone in the development of algorithmic approaches and lower bound techniques [21, 22, 24, 27, 43, 44]. The best known deterministic algorithm for MST with uncertainty is 2-competitive, and no deterministic algorithm can be better [24]. A randomized algorithm with competitive ratio 1.707 is known [43]. Further work considers the non-adaptive problem, which has a very different flavor [44].

In this paper, we assume that an algorithm has, for each edge e , access to a prediction $\bar{w}_e \in I_e$ for the unknown value w_e . For example, machine learning (ML) methods could be used to predict the value of an edge. Given the tremendous progress in artificial intelligence and ML in recent decades, we can expect that those predictions are of good accuracy, but there is no guarantee and the predictions might be completely wrong. This lack of provable performance guarantees for ML often causes concerns regarding how confident one can be that an ML algorithm will perform sufficiently well in all circumstances. We address the very natural question whether the availability of such (ML) predictions can be exploited by query algorithms for computing with explorable uncertainty. Ideally, an algorithm should perform very well if predictions are accurate, but even if they are arbitrarily wrong, the algorithm should not perform worse than an algorithm without access to predictions. To emphasize that the predictions can be wrong, we refer to them as *untrusted predictions*.

We note that the availability of both uncertainty intervals and untrusted predictions is natural in many scenarios. For example, the quality of links (measured using metrics such as throughput and reliability) in a wireless network often fluctuates over time within a certain interval, and ML methods can be used to predict the precise link quality based on time-series data of previous link quality measurements [1]. The actual quality of a link can be obtained via a new measurement. If we wish to build a minimum spanning tree using links that currently have the highest link quality and want to minimize the additional measurements needed, we arrive at an MST problem with uncertainty and untrusted predictions.

We study for the first time the combination of explorable uncertainty and untrusted predictions. Our work is inspired by the vibrant recent research trend of considering untrusted (ML) predictions in the context of *online* algorithms, a different uncertainty model where the input is revealed to an algorithm incrementally. Initial work on online caching problems [40] has initiated a vast growing line of research on caching [5, 49, 53], rent-or-buy problems [31, 48, 54], scheduling [4, 9, 38, 45, 48], graph problems [19, 37, 39] and many more.

We adopt the following notions introduced in [40, 48]: An algorithm is α -consistent if it is α -competitive when the predictions are correct, and it is β -robust if it is β -competitive no matter how wrong the predictions are. Furthermore, we are interested in a smooth transition between the case with correct predictions and the case with arbitrarily incorrect predictions. We aim for performance guarantees that degrade gracefully with increasing prediction error.

Given predicted values for the uncertainty intervals, it is tempting to simply run an optimal algorithm under the assumption that the predictions are correct. This is obviously optimal with respect to consistency, but might give arbitrarily bad solutions in the case when the predictions are faulty. Instead of blindly trusting the predictions, we need more sophisticated strategies to be robust against prediction errors. This requires new lower bounds on an optimal solution, new structural insights, and new algorithmic techniques.

Main results

In this work, we show that, in the setting of explorable uncertainty, it is in fact possible to exploit ML predictions of the uncertain values and improve the performance of a query strategy when the predictions are good, while at the same time guaranteeing a strong bound on the worst-case performance even when the predictions are arbitrarily bad.

We give algorithms for the MST problem with uncertainty that are parameterized by a hyperparameter γ that reflects the user's confidence in the accuracy of the predictor. For any integral $\gamma \geq 2$, we present a $(1 + \frac{1}{\gamma})$ -consistent and γ -robust algorithm, and show that this is the best possible trade-off between consistency and robustness. In particular, for $\gamma = 2$, we obtain a 2-robust, 1.5-consistent algorithm. It is worth noting that this algorithm achieves the improved competitive ratio of 1.5 for accurate predictions while maintaining the worst-case ratio of 2. This is in contrast to many learning-augmented online algorithms where the exploitation of predictions usually incurs an increase in the worst-case ratio (e.g., [6, 48]).

Our main result is a second and different algorithm with a more fine-grained performance analysis that obtains a guarantee that improves with the accuracy of the predictions. Very natural, simple error measures such as the number of inaccurate predictions or the ℓ_1 -norm of the difference between predictions and true values turn out to prohibit any reasonable error-dependency. Therefore, we propose an error measure, called *hop distance* k_h , that takes structural insights about uncertainty intervals into account and may also be useful for other problems in computing with uncertainty and untrusted predictions. We give a precise definition of this error measure later. We also show in the full version [20] that the predictions are efficiently PAC-learnable with respect to k_h . Our main result is a learning-augmented algorithm with a competitive ratio with a linear error-dependency $\min\{(1 + \frac{1}{\gamma}) + \frac{5 \cdot k_h}{|\text{OPT}|}, \gamma + 1\}$, for any integral $\gamma \geq 2$. All our algorithms have polynomial running-times. We describe our techniques and highlight their novelty in the following section.

The integrality requirement for γ comes from using γ to determine set sizes and can be removed by randomization at the cost of a slightly worse consistency guarantee; for a proof we refer to the full version.

Further related work

There is a long history of research on the tradeoff between exploration and exploitation when coping with uncertainty in the input data. Often, stochastic models are assumed, e.g., in work on multi-armed bandits [16, 28, 52], Weitzman's Pandora's box [55], and query-variants of combinatorial optimization problems; see, e.g., [32, 41, 51] and many more. In our work, we assume no knowledge of stochastic information and aim for robust algorithms that perform well even in a worst case.

The line of research on explorable uncertainty has been initiated by Kahan [35] in the context of selection problems. Subsequent work addressed finding the k -th smallest value in a set of uncertainty intervals [26, 33], caching problems [47], computing a function value [36], sorting [34], and classical combinatorial optimization problems. Some of the major aforementioned results on the MST problem under explorable uncertainty have been extended to general matroids [23, 43, 44]. Further problems that have been studied are the shortest path problem [25], the knapsack problem [29] and scheduling problems [2, 3, 7, 10, 18]. Although optimization under explorable uncertainty has been studied mainly in an adversarial model, recently first results have been obtained for stochastic variants for sorting [17] and selection type problems (hypergraph orientation) [11].

There is a significant body of work on computing in models where information about a hidden object can be accessed only via queries. The hidden object can, for example, be a function, a matrix, or a graph. In the graph context, property testing [30] has been studied extensively and there are many more works, see [8, 12, 13, 42, 46, 50]. The bounds on the number of queries made by an algorithm are typically absolute (as a function of the input) and the resulting correctness guarantees are probabilistic. This is very different from our work, where we aim for a comparison to the minimum number of queries needed for the given graph.

2 Overview of techniques and definition of error measure

We assume that each uncertainty interval is either *open*, $I_e = (L_e, U_e)$, or *trivial*, $I_e = \{w_e\}$, and we refer to edge e as *non-trivial* or *trivial*, respectively; a standard assumption to avoid a simple lower bound of $|E|$ on the competitive ratio [24, 33].

Before we give an overview of the used techniques, we formally define feasible and optimal query sets. We say that a query set $Q \subseteq E$ is *feasible* if there exists a set of edges $T \subseteq E$ such that T is an MST for the true values w_e of all $e \in Q$ and *every possible* combination of edge weights in I_e for the unqueried edges $e \in E \setminus Q$. That is, querying a feasible query set Q must give us sufficient information to identify a spanning tree T that is an MST for the true values no matter what the true values of the unqueried edges $E \setminus Q$ actually are. We call a feasible query set Q *optimal* if it has minimum cardinality $|Q|$ among all feasible query sets. Thus, the optimal solution depends only on the true values and not on the predicted values.

As Erlebach and Hoffmann [21] give a polynomial-time algorithm that computes an optimal query set under the assumption that all query results are known upfront, we can use their algorithm to compute the optimal query set under the assumption that all predicted values match the actual edge weights and query the computed set in an arbitrary order. If the predicted values are indeed correct, this yields a 1-consistent algorithm. However, such an algorithm may have an arbitrarily bad performance if the predictions are incorrect. Similarly, the known deterministic 2-competitive algorithm for the MST problem with uncertainty (without predictions) [24] is 2-robust and 2-consistent. The known lower bound of 2 rules out any robustness factor less than 2. It builds on the following simple example with two intersecting intervals I_a, I_b that are candidates for the largest edge weight in a cycle. No matter which interval a deterministic algorithm queries first, say I_a , the realized value could be $w_a \in I_a \cap I_b$, which requires a second query. If the adversary chooses $w_b \notin I_a \cap I_b$, querying just I_b would have been sufficient to identify the interval with larger true value. See also [24, Example 3.8] and [43, Section 3] for an illustration of the lower bound example.

Algorithm overview

We aim for $(1 + \frac{1}{\gamma})$ -consistent and γ -robust algorithms for each integral $\gamma \geq 2$. The algorithm proceeds in two phases: The first phase runs as long as there are prediction mandatory edges, i.e., edges that must be contained in every feasible query set under the assumption that the predictions are correct; we later give a formal characterization of such edges. In this phase, we exploit the existence of those edges and their properties to execute queries with strong local guarantees, i.e., each feasible query set contains a large portion of our queries. For the second phase, we observe and exploit that the absence of prediction mandatory queries implies that the predicted optimal solution is a minimum vertex cover in a bipartite auxiliary graph. The challenge here is that the auxiliary graph can change with each wrong prediction. To obtain an error-dependent guarantee (our error measure k_h is discussed below) we need to adaptively query a dynamically changing minimum vertex cover.

Novel techniques

During the first phase, we generalize the classical witness set analysis. In computing with explorable uncertainty, the concept of *witness sets* is crucial for comparing the query set of an algorithm with an optimal solution (a way of lower-bounding). A witness set [15] is a set of elements for which we can guarantee that any feasible solution must query at least *one* of these elements. Known algorithms for the MST problem without predictions [24, 43] essentially follow the algorithms of Kruskal or Prim and only identify witness sets of size 2 in the cycle or cut that is currently under consideration. Querying disjoint witness sets of size 2 (witness pairs) ensures 2-robustness but does not lead to an improved consistency.

In our first phase, we extend this concept by considering *strengthened* witness sets of three elements such that any feasible query set must contain at least two of them. Since we cannot always find strengthened witness sets based on structural properties alone (otherwise, there would be a 1.5-competitive algorithm for the problem without predictions), we identify such sets under the assumption that the predictions are correct. Even after identifying such elements, the algorithm needs to query them in a careful order: if the predictions are wrong, we lose the guarantee on the elements, and querying all of them might violate the robustness. In order to identify strengthened witness sets, we provide new, more global criteria to identify additional witness sets (of size two) beyond the current cycle or cut. These new ways to identify witness sets are a major contribution that may be of independent interest regardless of predictions. During the first phase, we repeatedly query $\gamma - 2$ prediction mandatory edges together with a strengthened witness set, which ensures $(1 + \frac{1}{\gamma})$ -consistency. We query the elements in a carefully selected order while adjusting for errors to ensure γ -robustness.

For the second phase, we observe that the predicted optimal solution of the remaining instance is a minimum vertex cover VC in a bipartite auxiliary graph representing the structure of *potential* witness pairs (edges of the input graph correspond to vertices of the auxiliary graph). For instances with this property, we aim for 1-consistency and 2-robustness; the best-possible trade-off for such instances. If the predictions are correct, each edge of the auxiliary graph is a witness pair. However, if a prediction error is observed when a vertex of VC is queried, the auxiliary graph changes. This means that some edges of the original auxiliary graph are not actually witness pairs. Indeed, the size of a minimum vertex cover can increase and decrease and does not constitute a lower bound on $|\text{OPT}|$; we refer to the full version for an example.

If we only aim for consistency and robustness, we can circumvent this problem by selecting a distinct matching partner $h(e) \notin VC$ for each $e \in VC$ applying *König-Egerváry's* Theorem (duality of maximum matchings and minimum vertex covers in bipartite graphs, see e.g. [14]).

By querying the elements of VC in a carefully chosen order until a prediction error is observed for the first time, we can guarantee that $\{e, h(e)\}$ is a witness set for each $e \in VC$ that is already queried. In the case of an error, this allows us to extend the previously queried elements to disjoint witness pairs to guarantee 2-robustness. Then, we can switch to an arbitrary (prediction-oblivious) 2-competitive algorithm for the remaining queries.

If we additionally aim for an error-sensitive guarantee, however, handling the dynamic changes to the auxiliary graph, its minimum vertex cover VC and matching h requires substantial additional work, and we see overcoming this challenge as our main contribution. In particular, querying the partner $h(e)$ of each already queried $e \in VC$ in case of an error might be too expensive for the error-dependent guarantee. However, if we do not query these partners, the changed instance still depends on them, and if we charge against such a partner multiple times, we can lose the robustness. Our solution is based on an elaborate charging/counting scheme and involves:

- keeping track of matching partners of already queried elements of VC ;
- updating the matching and VC using an augmenting path method to bound the number of elements that are charged against multiple times in relation to the prediction error;
- and querying the partners of previously queried edges (and their new matching partners) as soon as they become endpoints of a newly matched edge, in order to prevent dependencies between the (only partially queried) witness sets of previously queried edges.

The error-sensitive algorithm achieves a competitive ratio of $1 + \frac{1}{\gamma} + \frac{5k_h}{|\text{OPT}|}$, at the price of a slightly increased robustness of $\gamma + 1$ instead of γ .

Hop distance as error metric

When we aim for a fine-grained performance analysis giving guarantees that depend on the quality of predictions, we need a metric to measure the prediction error. A very natural, simple error measure is the number of inaccurate predictions $k_{\#} = |\{e \in E \mid w_e \neq \bar{w}_e\}|$. However, we can show that even for $k_{\#} = 1$ the competitive ratio cannot be better than the known bound of 2 (see Lemma 19 in the full version). The reason for the weakness of this measure is that it completely ignores the interleaving structure of intervals. Similarly, an ℓ_1 error metric such as $\sum_{e \in E} |w_e - \bar{w}_e|$ would not be meaningful because only the *order* of the values and the interval endpoints matters for our problems.

We propose a refined error measure, which we call *hop distance*. It is very intuitive even though it requires some technical care to make it precise. If we consider only a single predicted value \bar{w}_e for some $e \in E$, then, in a sense, this value predicts the relation of the true value w_e to the intervals of edges $e' \in E \setminus \{e\}$. In particular, w.r.t. a fixed $e' \in E \setminus \{e\}$, the value \bar{w}_e predicts whether w_e is left of $I_{e'}$ ($\bar{w}_e \leq L_{e'}$), right of $I_{e'}$ ($\bar{w}_e \geq U_{e'}$), or contained in $I_{e'}$ ($L_{e'} < \bar{w}_e < U_{e'}$). Interpreting the prediction \bar{w}_e in this way, the prediction is “wrong” (w.r.t. a fixed $e' \in E \setminus \{e\}$) if the predicted relation of the true value w_e to interval $I_{e'}$ is not actually true, e.g., w_e is predicted to be left of $I_{e'}$ ($\bar{w}_e \leq L_{e'}$) but the actual w_e is either contained in or right of $I_{e'}$ ($w_e > L_{e'}$). Formally, we define the function $k_{e'}(e)$ that indicates whether the predicted relation of w_e to $I_{e'}$ is true ($k_{e'}(e) = 0$) or not ($k_{e'}(e) = 1$). With the prediction error $k^+(e)$ for a single $e \in E$, we want to capture the number of relations between w_e and intervals $I_{e'}$ with $e' \in E \setminus \{e\}$ that are not accurately predicted. Thus, we define $k^+(e) = \sum_{e' \in E \setminus \{e\}} k_{e'}(e)$. For a set of edges $E' \subseteq E$, we define $k^+(E') = \sum_{e \in E'} k^+(e)$. Consequently, with the error for the complete instance we want to capture the total number of wrongly predicted relations and, therefore, define it by $k_h = k^+(E)$. We call this error measure k_h the *hop distance*; see Figure 1 for an illustration.

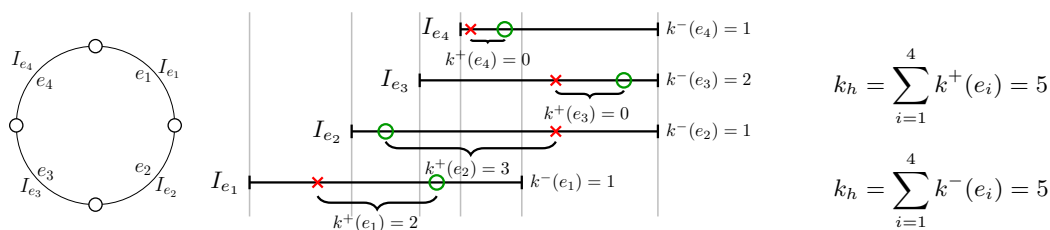


Figure 1 Example of a single cycle (left) with uncertain edge weights from intersecting intervals $I_{e_1}, I_{e_2}, I_{e_3}, I_{e_4}$ (middle). Circles illustrate true values and crosses illustrate the predicted values.

Symmetrically, we can define $k^-(e) = \sum_{e' \in E \setminus \{e\}} k_e(e')$ and $k^-(E') = \sum_{e \in E'} k^-(e)$ for subset $E' \subseteq E$. Then $k^+(E) = k_h = k^-(E)$ follows by reordering the summations.

3 Structural results

In this section, we introduce some known concepts and prove new structural results on MST under explorable uncertainty, which we use later to design learning-augmented algorithms.

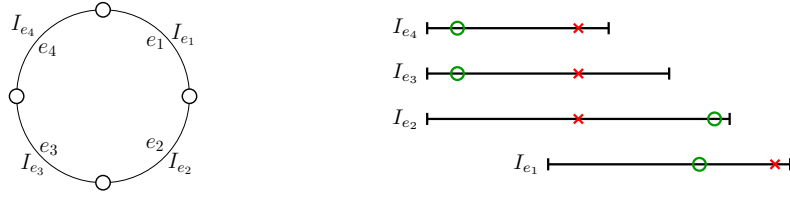
Witness sets and mandatory queries

Witness sets are the key to the analysis of query algorithms. They allow for a comparison of an algorithm’s query set to an optimal solution. A subset $W \subseteq E$ is a *witness set* if $W \cap Q \neq \emptyset$ for all feasible query sets Q . An important special case are witness sets of cardinality one, i.e., edges that are part of every feasible query set. We call such edges *mandatory*. Similarly, we call edges that are mandatory under the assumption that the predictions are correct *prediction mandatory*.

For an example, consider Figure 2. In the example, we see the uncertainty intervals, predicted values and true values of four edges that form a simple cycle. We can observe that both e_1 and e_2 are mandatory for this example. To see this, assume that e_1 is not mandatory. Then, there must be a feasible query set Q with $e_1 \notin Q$ for the instance, which implies that $Q = \{e_2, e_3, e_4\}$ must be feasible. But even after querying Q to reveal the true values of e_2, e_3 and e_4 , it still depends on the still unknown true value of e_1 whether there exists an MST T with $e_1 \in T$ (only if $w_{e_1} \leq w_{e_2}$) and/or $e_1 \notin T$ (only if $w_{e_2} \leq w_{e_1}$). Even after querying Q there is no spanning tree T that is an MST for each possible edge weight in I_{e_1} of the unqueried edge e_1 and, thus, Q is not feasible. This implies that e_1 is mandatory, and we can argue analogously that e_2 is mandatory as well.

To argue whether an edge is prediction mandatory, on the other hand, we assume that all queries reveal the predicted values as true values. Under this assumption, a query to e_1 in the example would reveal a value that is larger than all upper limits U_{e_i} with $i \in \{2, 3, 4\}$, which implies that e_1 cannot be part of any MST and that $T = \{e_2, e_3, e_4\}$ is an MST no matter what the true values of e_2, e_3 and e_4 actually are. Therefore, under the assumption that all predicted values match the true values, $Q = \{e_1\}$ is a feasible query set and, thus, e_2 is not prediction mandatory despite being mandatory. However, we can use a similar argumentation as above to argue that e_1 is also prediction mandatory.

We continue by giving properties that allow us to identify (prediction) mandatory edges. To that end, let the *lower limit tree* $T_L \subseteq E$ be an MST for values w^L with $w_e^L = L_e + \epsilon$ for an infinitesimally small $\epsilon > 0$. Analogously, let the *upper limit tree* T_U be an MST for values w^U with $w_e^U = U_e - \epsilon$. This concept has been introduced in [43] to identify mandatory



■ **Figure 2** Example of a single cycle (left) with uncertain edge weights from intersecting intervals $I_{e_1}, I_{e_2}, I_{e_3}, I_{e_4}$ (right). Circles illustrate true values and crosses illustrate the predicted values. For the example, $\{e_1, e_2\}$ is the set of all mandatory edges and $\{e_1\}$ is the set of all prediction mandatory edges.

queries; it is shown that every non-trivial $e \in T_L \setminus T_U$ is mandatory. Thus, we may repeatedly query edges in $T_L \setminus T_U$ until $T_L = T_U$ without worsening the competitive ratio. We can extend this preprocessing to achieve uniqueness for T_L and T_U and, thus, may assume unique $T_L = T_U$.

► **Lemma 1.** *By querying only mandatory elements we can obtain an instance with $T_L = T_U$ such that T_L and T_U are the unique lower limit tree and upper limit tree, respectively.*

Given an instance with unique $T_L = T_U$, we observe that each $e \in T_L$ that is not part of the MST for the true values is mandatory. Similarly, each $e \notin T_L$ that is part of the MST for the true values is mandatory as well. A stronger version of this observation is as follows.

► **Lemma 2.** *Let G be an instance with unique $T_L = T_U$ and let G' be an instance with unique $T'_L = T'_U$ obtained from G by querying set Q , then $e \in T_L \Delta T'_L = (T_L \setminus T'_L) \cup (T'_L \setminus T_L)$ implies $e \in Q$.*

Next we establish a relation between the set $E_M \subseteq E$ of mandatory queries, the set $E_P \subseteq E$ of prediction mandatory queries, and the hop distance k_h .

► **Lemma 3.** *Consider a given problem instance $G = (V, E)$ with predicted values \bar{w} . Each $e \in E_M \Delta E_P$ satisfies $k^-(e) \geq 1$. Consequently, $k_h \geq |E_M \Delta E_P|$.*

For a formal proof, we refer to the full version. Intuitively, if $e \in E_P \setminus E_M$, then it is possible to solve the instance without querying e . Thus, the relation of the true values $w_{e'}$ with $e' \in E \setminus \{e\}$ to interval I_e must be such that querying $E \setminus \{e\}$ allows us to verify that e is either part or not part of the MST. If the predicted relations of the true values $w_{e'}$ with $e' \in E \setminus \{e\}$ to interval I_e were the same, then querying $E \setminus \{e\}$ would also allow us to verify that e is either part or not part of the MST. Thus, the predicted relation of at least one $w_{e'}$ to interval I_e must be wrong. We can argue analogously for $e \in E_M \setminus E_P$.

Identifying witness sets

We introduce new structural properties to identify witness sets. Existing algorithms for MST under uncertainty [24, 43] essentially follow the algorithms of Kruskal or Prim, and only identify witness sets in the cycle or cut that is currently under consideration. Let f_1, \dots, f_l denote the edges in $E \setminus T_L$ ordered by non-decreasing lower limit. Then, C_i with $i \in \{1, \dots, l\}$ denotes the unique cycle in $T_L \cup \{f_i\}$.

For each $e \in T_L$, let X_e denote the set of edges in the cut of the two connected components of $T_L \setminus \{e\}$. Existing algorithms for MST under explorable uncertainty repeatedly consider (the changing) C_1 or X_e , where e is the edge in T_L with maximum upper limit, and identify

the maximum or minimum edge in the cycle or cut by querying witness sets of size two, until the problem is solved. For our algorithms, we need to identify witness sets in cycles $C_i \neq C_1$ and cuts $X_{e'} \neq X_e$.

► **Lemma 4.** *Consider cycle C_i with $i \in \{1, \dots, l\}$. Let $l_i \in C_i \setminus \{f_i\}$ such that $I_{l_i} \cap I_{f_i} \neq \emptyset$ and l_i has the largest upper limit in $C_i \setminus \{f_i\}$, then $\{f_i, l_i\}$ is a witness set. If $w_{f_i} \in I_{l_i}$, then l_i is mandatory.*

Characterization of prediction mandatory free instances

We say an instance is *prediction mandatory free* if it contains no prediction mandatory elements. A key part of our algorithms is to transform instances into prediction mandatory free instances while maintaining a competitive ratio that allows us to achieve the optimal consistency and robustness trade-off overall. We give the following characterization of prediction mandatory free instances, (cf. Figure 3). Then, we show that prediction mandatory free instances remain so as long as we ensure $T_L = T_U$.



■ **Figure 3** Intervals in a prediction mandatory free cycle with predictions indicated as red crosses.

► **Lemma 5.** *An instance G is prediction mandatory free if and only if $\bar{w}_{f_i} \geq U_e$ and $\bar{w}_e \leq L_{f_i}$ holds for each $e \in C_i \setminus \{f_i\}$ and each cycle C_i with $i \in \{1, \dots, l\}$. Once an instance is prediction mandatory free, it remains so even if we query further elements, as long as we maintain unique $T_L = T_U$.*

Making instances prediction mandatory free

In the full version, we give a powerful preprocessing algorithm that transforms arbitrary instances into prediction mandatory free instances.

► **Theorem 6.** *There is an algorithm that makes a given instance prediction mandatory free and satisfies $|\text{ALG}| \leq \min\{(1 + \frac{1}{\gamma}) \cdot (|\text{ALG} \cup D| \cap \text{OPT}| + k^+(\text{ALG}) + k^-(\text{ALG})), \gamma \cdot (|\text{ALG} \cup D| \cap \text{OPT}| + \gamma - 2)\}$ for the set of edges ALG queried by the algorithm and a set $D \subseteq E \setminus \text{ALG}$ of unqueried edges that do not occur in the remaining instance after executing the algorithm.*

The set D are edges that, even without being queried by the algorithm, are proven to be maximal in a cycle or minimal in a cut. Thus, they can be deleted or contracted w.l.o.g. and do not exist in the instance remaining *after* executing the preprocessing algorithm. This is an important property as it means that the remaining instance is independent of D and ALG (as all elements of ALG are already queried). Since the theorem compares $|\text{ALG}|$ with $|\text{ALG} \cup D| \cap \text{OPT}|$ instead of just $|\text{OPT}|$, this allows us to combine the given guarantee with the guarantees of dedicated algorithms for prediction mandatory free instances. However, we have to be careful with the additive term $\gamma - 2$, but we will see that we can charge this term against the improved robustness of our algorithms for prediction mandatory free instances.

4 An algorithm with optimal consistency and robustness trade-off

We give a bound on the best achievable tradeoff between consistency and robustness.

► **Theorem 7.** *Let $\beta \geq 2$ be a fixed integer. For the MST problem under explorable uncertainty with predictions, there is no deterministic β -robust algorithm that is α -consistent for $\alpha < 1 + \frac{1}{\beta}$. And vice versa, no deterministic α -consistent algorithm, with $\alpha > 1$, is β -robust for $\beta < \max\{\frac{1}{\alpha-1}, 2\}$.*

The main result of this section is an optimal algorithm w.r.t. this tradeoff bound.

► **Theorem 8.** *For every integer $\gamma \geq 2$, there exists a $(1 + \frac{1}{\gamma})$ -consistent and γ -robust algorithm for the MST problem under explorable uncertainty with predictions.*

To show this result, we design an algorithm for prediction mandatory free instances with unique $T_L = T_U$. We run it after the preprocessing algorithm which obtains such special instance with the query guarantee in Theorem 6. Our new algorithm achieves the optimal trade-off.

► **Theorem 9.** *There exists a 1-consistent and 2-robust algorithm for prediction mandatory free instances with unique $T_L = T_U$ of the MST problem under explorable uncertainty with predictions.*

In a prediction mandatory free instance $G = (V, E)$, each $f_i \in E \setminus T_L$ is predicted to be maximal on cycle C_i , and each $l \in T_L$ is predicted to be minimal in X_l (cf. Lemma 5). If these predictions are correct, then T_L is an MST and the optimal query set is a minimum vertex cover in a bipartite graph $\bar{G} = (\bar{V}, \bar{E})$ with $\bar{V} = E$ (excluding trivial edges) and $\bar{E} = \{\{f_i, e\} \mid i \in \{1, \dots, l\}, e \in C_i \setminus \{f_i\} \text{ and } I_e \cap I_{f_i} \neq \emptyset\}$ [21, 43]. We refer to \bar{G} as the *vertex cover instance*. Note that if a query reveals that an f_i is not maximal on C_i or an $l \in T_L$ is not minimal in X_l , then the vertex cover instance changes. Let VC be a minimum vertex cover of \bar{G} . Non-adaptively querying VC ensures 1-consistency but might lead to an arbitrarily bad robustness. Indeed, the size of a minimum vertex cover can increase and decrease drastically as we show in the full version. Thus, the algorithm has to be more adaptive.

The idea of the algorithm (cf. Algorithm 1) is to sequentially query each $e \in VC$ and charge for querying e by a distinct non-queried element $h(e)$ such that $\{e, h(e)\}$ is a witness set. Querying exactly one element per distinct witness set implies optimality. To identify $h(e)$ for each element $e \in VC$, we use the fact that *König-Egerváry's* Theorem (e.g., [14]) on the duality between minimum vertex covers and maximum matchings in bipartite graphs implies that there is a matching h that maps each $e \in VC$ to a distinct $e' \notin VC$. While the sets $\{e, h(e)\}$ with $e \in VC$ in general are not witness sets, querying VC in a specific order until the vertex cover instance changes guarantees that $\{e, h(e)\}$ is a witness set for each already queried e . The algorithm queries in this order until it detects a wrong prediction or solves the problem. If it finds a wrong prediction, it queries the partner $h(e)$ of each already queried edge e , and continues to solve the problem with a 2-competitive algorithm (e.g., [24, 43]). The following lemma specifies the order in which the algorithm queries VC .

► **Lemma 10.** *Let l'_1, \dots, l'_k be the edges in $VC \cap T_L$ ordered by non-increasing upper limit and let d be such that the true value of each l'_i with $i < d$ is minimal in cut $X_{l'_i}$, then $\{l'_i, h(l'_i)\}$ is a witness set for each $i \leq d$. Let f'_1, \dots, f'_g be the edges in $VC \setminus T_L$ ordered by non-decreasing lower limit and let b be such that the true value of each f'_i with $i < b$ is maximal in cycle $C_{f'_i}$, then $\{f'_i, h(f'_i)\}$ is a witness set for each $i \leq b$.*

■ **Algorithm 1** 1-consistent and 2-robust algorithm for prediction mandatory free instances.

Input: Prediction mandatory free graph $G = (V, E)$ with unique $T_L = T_U$.

- 1 Compute maximum matching h and minimum vertex cover VC for \tilde{G} ;
- 2 Set $W = \emptyset$, and let f'_1, \dots, f'_g and l'_1, \dots, l'_k be as described in Lemma 10;
- 3 **for** e chosen sequentially from the ordered list $f'_1, \dots, f'_g, l'_1, \dots, l'_k$ **do**
- 4 Query e and add $h(e)$ to W ;
- 5 **if** $k^+(e) \neq 0$ **then** query set W and solve the instance with a 2-competitive algorithm;

Proof. Here, we show the first statement and refer to the full version for the proof of the second statement. Consider an arbitrary l'_i and $h(l'_i)$ with $i \leq d$. By definition of h , the edge $h(l'_i)$ is not part of the lower limit tree. Consider $C_{h(l'_i)}$, i.e., the cycle in $T_L \cup \{h(l'_i)\}$, then we claim that $C_{h(l'_i)}$ only contains $h(l'_i)$ and edges in $\{l'_1, \dots, l'_k\}$ (and possibly irrelevant edges that do not intersect $I_{h(l'_i)}$). To see this, recall that $l'_i \in VC$, by definition of h , implies $h(l'_i) \notin VC$. For VC to be a vertex cover, each $e \in C_{h(l'_i)} \setminus \{h(l'_i)\}$ must either be in VC or not intersect $h(l'_i)$. Consider the relaxed instance where the true values for each l'_j with $j < d$ and $j \neq i$ are already known. By assumption each such l'_j is minimal in its cut $X_{l'_j}$. Thus, we can w.l.o.g. contract each such edge. It follows that in the relaxed instance l'_i has the highest upper limit in $C_{h(l'_i)} \setminus \{h(l'_i)\}$. According to Lemma 4, $\{l'_i, h(l'_i)\}$ is a witness set. ◀

Proof of Theorem 9. We first show 1-consistency. Assume that all predictions are correct, then VC is an optimal query set and $k^+(e) = 0$ holds for all $e \in E$. It follows that Line 5 never executes queries and the algorithm queries exactly VC . This implies 1-consistency.

Further, if the algorithm never queries in Line 5, then the consistency analysis implies 1-robustness. Suppose Line 5 executes queries. Let Q_1 denote the set of edges that are queried before the queries of Line 5 and let $Q_2 = \{h(e) \mid e \in Q_1\}$. Then Q_2 corresponds to the set W as queried in Line 5. By Lemma 10, each $\{e, h(e)\}$ with $e \in Q_1$ is a witness set. Further, the sets $\{e, h(e)\}$ are pairwise disjoint. Thus, $|Q_1 \cup Q_2| \leq 2 \cdot |\text{OPT} \cap (Q_1 \cup Q_2)|$. Apart from $Q_1 \cup Q_2$, the algorithm queries a set Q_3 in Line 5 to solve the remaining instance with a 2-competitive algorithm. So, $|Q_3| \leq 2 \cdot |\text{OPT} \setminus (Q_1 \cup Q_2)|$ and, adding up the inequalities, $|\text{ALG}| \leq 2 \cdot |\text{OPT}|$. ◀

A careful combination of Theorems 6 and 9 implies Theorem 8. Full proof in the full version.

5 An error-sensitive algorithm

In this section, we extend the algorithm of Section 4 to obtain error sensitivity. First, we note that $k_{\#} = 0$ implies $k_h = 0$, so Theorem 7 implies that no algorithm can simultaneously have competitive ratio better than $1 + \frac{1}{\beta}$ if $k_h = 0$ and β for arbitrary k_h . In addition, we can give the following lower bound on the competitive ratio as a function of k_h .

► **Theorem 11.** *Any deterministic algorithm for MST under explorable uncertainty with predictions has a competitive ratio $\rho \geq \min\{1 + \frac{k_h}{|\text{OPT}|}, 2\}$, even for edge disjoint prediction mandatory free cycles.*

Again, we design an algorithm for prediction mandatory free instances with unique $T_L = T_U$ and use it in combination with the preprocessing algorithm (Theorem 6) to prove the following.

► **Theorem 12.** *For every integer $\gamma \geq 2$, there exists a $\min\{1 + \frac{1}{\gamma} + \frac{5 \cdot k_h}{|\text{OPT}|}, \gamma + 1\}$ -competitive algorithm for the MST problem under explorable uncertainty with predictions.*

We actually show a robustness of $\max\{3, \gamma + \frac{1}{|\text{OPT}|}\}$ which might be smaller than $\gamma + 1$. Our algorithm for prediction mandatory free instances asymptotically matches the error-dependent guarantee of Theorem 11 at the cost of a slightly worse robustness.

► **Theorem 13.** *There exists a $\min\{1 + \frac{5 \cdot k_h}{|\text{OPT}|}, 3\}$ -competitive algorithm for prediction mandatory free instances with unique $T_L = T_U$ of the MST problem under explorable uncertainty with predictions.*

We follow the same strategy as before. However, Algorithm 1 just executes a 2-competitive algorithm once it detects an error. This is sufficient to achieve the optimal trade-off as we, if an error occurs, only have to guarantee 2-competitiveness. To obtain an error-sensitive guarantee however, we have to ensure both, $|\text{ALG}| \leq 3 \cdot |\text{OPT}|$ and $|\text{ALG}| \leq \text{OPT} + 5 \cdot k_h$ even if errors occur. Further, we might not be able to afford queries to the complete set W (Algorithm 1, Line 5) in the case of an error as this might violate $|\text{ALG}| \leq \text{OPT} + 5 \cdot k_h$.

We adjust the algorithm to query elements of f'_1, \dots, f'_g and l'_1, \dots, l'_k as described in Lemma 10 not only until an error occurs but until the vertex cover instance changes. That is, until some f_i that at the beginning of the iteration is not part of T_L becomes part of the lower limit tree, or some l_i that at the beginning of the iteration is part of T_L is not part of the lower limit tree anymore. Once the instance changes, we recompute both, the bipartite graph \bar{G} as well as the matching h and minimum vertex cover VC for \bar{G} . Instead of querying the complete set W , we only query the elements of W that occur in the recomputed matching, as well as the new matching partners of those elements. And instead of switching to a 2-competitive algorithm, we restart the algorithm with the recomputed matching and vertex cover. Algorithm 2 formalizes this approach. In the algorithm, h denotes a matching that matches each $e \in VC$ to a distinct $h(e) \notin VC$; we use the notation $\{e, e'\} \in h$ to indicate that h matches e and e' . For a subset $U \subseteq VC$ let $h(U) = \{h(e) \mid e \in U\}$. For technical reasons, the algorithm does not recompute an arbitrary matching h but follows the approach of Lines 10 and 11. Intuitively, an arbitrary maximum matching h might contain too many elements of W , which would lead to too many additional queries.

Let ALG denote the queries of Algorithm 2 on a prediction mandatory free instance with unique $T_L = T_U$. We show Theorem 13 by proving $\text{ALG} \leq \text{OPT} + 5 \cdot k_h$ and $\text{ALG} \leq 3 \cdot \text{OPT}$.

Before proving the two inequalities, we state some key observations about the algorithm. We argue that an element e' can never be part of a partial matching \bar{h} in an execution of Line 10 *after* it is added to set W . Recall that the vertex cover instances only contain non-trivial elements. Thus, if an element e is queried in Line 5 and the current partner $e' = h(e)$ is added to set W , then the vertex cover instance at the next execution of Line 10 does not contain the edge $\{e, e'\}$ and, therefore, e' is not part of the partial matching \bar{h} of that line. As long as e' is not added to the matching by Line 11, it, by definition, can never be part of a partial matching \bar{h} in an execution of Line 10. As soon as the element e' is added to the matching in some execution of Line 11, it is queried in the following execution of Line 12. Therefore, e' can also not be part of a partial matching \bar{h} in an execution of Line 10 after it is added to the matching again. This leads to the following observation.

► **Observation 14.** *An element e' can never be part of a partial matching \bar{h} in an execution of Line 10 after it is added to set W . Once e' is added to the matching again in an execution of Line 11, it is queried directly afterwards in Line 12, and cannot occur in Line 5 anymore.*

■ **Algorithm 2** Error-sensitive algorithm for prediction mandatory free instances.

Input: Prediction mandatory free graph $G = (V, E)$ with unique $T_L = T_U$.

- 1 Compute maximum matching h and minimum vertex cover VC for \bar{G} and set $W = \emptyset$;
- 2 Let f'_1, \dots, f'_g and l'_1, \dots, l'_k be as described in Lemma 10 for VC and h ;
- 3 $L \leftarrow T_L, N \leftarrow E \setminus T_L$; /* recompute the actual T_L after each query */
- 4 **for** e chosen sequentially from the ordered list $f'_1, \dots, f'_g, l'_1, \dots, l'_k$ **do**
- 5 If e is non-trivial, i.e., has not been queried yet, query e and add $h(e)$ to W ;
- 6 Apply Lemma 1 to ensure unique $T_L = T_U$. For each query e' , if
 $\exists a$ s.t. $\{e', a\} \in h$, query a ;
- 7 Let $\bar{G}' = (\bar{V}', \bar{E}')$ be the vertex cover instance for the current instance;
- 8 **if** some $e' \in L$ is not in T_L or some $e' \in N$ is in T_L **then**
- 9 **repeat**
- 10 Let $\bar{G} = \bar{G}'$ and $\bar{h} = \{\{e', e''\} \in h \mid \{e', e''\} \in \bar{E}'\}$;
- 11 Compute h and VC by completing \bar{h} with an augmenting path algorithm;
- 12 Query $R = (VC \cup h(VC)) \cap (W \cup \{e' \mid \exists e \in W \text{ with } \{e, e'\} \in h\})$;
- 13 Ensure unique $T_L = T_U$. For each query e' , if $\exists a$ s.t. $\{e', a\} \in h$, query a ;
- 14 Let $\bar{G}' = (\bar{V}', \bar{E}')$ be the vertex cover instance for the current instance;
- 15 **until** $R = \emptyset$;
- 16 Restart at Line 2. In particular, do *not* reset W ;

We first analyze the queries that are *not* executed in Line 12. Let $Q_1 \subseteq \text{ALG}$ denote the queries of Line 5. For each $e \in Q_1$ let $h^*(e)$ be the matching partner of e at the time it was queried, and let $h^*(Q_1) = \bigcup_{e \in Q_1} \{h^*(e)\}$. Finally, let Q_2 denote the queries of Lines 6 and 13 to elements of $h^*(Q_1)$, and let Q_3 denote the remaining queries of those lines.

► **Lemma 15.** $|Q_1 \cup Q_3 \cup h^*(Q_1)| \leq 2 \cdot |\text{OPT} \cap (Q_1 \cup Q_3 \cup h^*(Q_1))|$ and $|Q_1 \cup Q_2 \cup Q_3| \leq |\text{OPT} \cap (Q_1 \cup Q_3 \cup h^*(Q_1))| + k^-(Q_2 \cup Q_3)$.

Proof. First, consider Q_1 and $h^*(Q_1)$. By Lemma 5, the instance is prediction mandatory free at the beginning of each restart of the algorithm. By Lemma 10, each $\{e, h^*(e)\}$ with $e \in Q_1$ is a witness set. We claim that all such $\{e, h^*(e)\}$ are pairwise disjoint, which implies $|Q_1 \cup h^*(Q_1)| \leq 2 \cdot |\text{OPT} \cap (Q_1 \cup h^*(Q_1))|$. Otherwise, an element of $\{e, h^*(e)\}$ must occur a second time in Line 5 after e is queried and $h^*(e)$ is added to W . Thus, either e or $h^*(e)$ must become part of a recomputed matching in line 10. By Observation 14 and since e becomes trivial, this cannot happen.

Consider an $e \in Q_2 \subseteq h^*(Q_1)$ and let $e' \in Q_1$ with $h^*(e') = e$. Since $e' \in Q_1$, it was queried in Line 5. Observe that e must have been queried after e' , as otherwise either e' would not have been queried in Line 5 (but together with e in Line 6 or 13), or e would not have been the matching partner of e' when it was queried by Observation 14; both contradict $e' \in Q_1$ and $h^*(e') = e$. This and Observation 14 imply that, at the time e is queried, its current matching partner is either the trivial e' or it has no partner. So, e must have been queried because it was mandatory and not as the matching partner of a mandatory element. Thus, each query of Q_2 is mandatory but, by Lemma 5, not prediction mandatory at the beginning of the iteration in which it is queried. Therefore, Lemma 3 implies that all mandatory elements e of Q_2 have $k^-(e) \geq 1$. It follows $|Q_1 \cup Q_2| \leq |\text{OPT} \cap (Q_1 \cup h^*(Q_1))| + k^-(Q_2)$.

By the argument above, no element of Q_3 was queried as the matching partner to an element of $Q_2 \cup Q_1$. Thus, by Lemma 1 and the definition of the algorithm, at least half the elements of Q_3 are mandatory, and we have $|Q_3| \leq 2 \cdot |\text{OPT} \cap Q_3|$ (and $\frac{1}{2}|Q_3| \leq |\text{OPT} \cap Q_3|$), which implies $|Q_1 \cup Q_3 \cup h^*(Q_1)| \leq 2 \cdot |\text{OPT} \cap (Q_1 \cup Q_3 \cup h^*(Q_1))|$.

By the same argument as for Q_2 , all mandatory elements e of Q_3 have $k^-(e) \geq 1$. Thus, $k^-(Q_3) \geq \frac{1}{2} \cdot |Q_3|$. Combining $k^-(Q_3) \geq \frac{1}{2} \cdot |Q_3|$ and $\frac{1}{2}|Q_3| \leq |\text{OPT} \cap Q_3|$ implies $|Q_3| \leq |\text{OPT} \cap Q_3| + k^-(Q_3)$. So, $|Q_1 \cup Q_2 \cup Q_3| \leq |\text{OPT} \cap (Q_1 \cup Q_2 \cup Q_3)| + k^-(Q_2 \cup Q_3)$. ◀

The first part of Lemma 15 captures all queries outside of Line 12 and all queries of Line 12 to elements of $W = h^*(Q_1)$. Let Q'_4 be the remaining queries of Line 12. By definition of the algorithm, $|Q'_4| \leq |W|$. Since $|W| \leq |\text{OPT}|$, we can conclude the next lemma.

► **Lemma 16.** $|\text{ALG}| \leq 3 \cdot |\text{OPT}|$.

Next, we show $|\text{ALG}| \leq |\text{OPT}| + 5 \cdot k_h$. Lemma 15 implies $|Q_1 \cup Q_2 \cup Q_3| \leq |\text{OPT} \cap (Q_1 \cup Q_2 \cup Q_3)| + k^-(Q_2 \cup Q_3)$. Hence, it remains to upper bound $|Q_4|$ with $Q_4 = \text{ALG} \setminus (Q_1 \cup Q_2 \cup Q_3)$ by $4 \cdot k_h$. By definition, Q_4 only contains edges that are queried in Line 12. Thus, at least half the queries of Q_4 are elements of W that are part of the matching h . By Observation 14, no element of W is part of the partial matching \bar{h} in Line 10. Hence, in each execution of Line 12, at least half the queries are not part of \bar{h} in Line 10 but added to h in Line 11. Our goal is to bound the number of such elements.

We start with some definitions. Define G_j as the problem instance at the j 'th execution of Line 11, and let G_0 denote the initial problem instance. For each G_j , let $\bar{G}_j = (\bar{V}_j, \bar{E}_j)$, T_L^j and T_U^j denote the corresponding vertex cover instance and lower and upper limit trees. Observe that each G_j has unique $T_L^j = T_U^j$, and, by Lemma 5, is prediction mandatory free. Let Y_j denote the set of queries made by the algorithm to transform instance G_{j-1} into instance G_j . We partition Q_4 into subsets S_j , where S_j contains the edges of Q_4 that are queried in the j 'th execution of Line 12. We claim $|S_j| \leq 4 \cdot k^+(Y_j)$ for each j , which implies $|Q_4| \leq \sum_j |S_j| \leq 4 \cdot \sum_j k^+(Y_j) \leq 4 \cdot k_h$. To show the claim, we rely on the following lemma.

► **Lemma 17.** *Let l, f be non-trivial edges in G_j such that $\{l, f\} \in \bar{E}_{j-1} \Delta \bar{E}_j$, then $k^-(l), k^-(f) \geq 1$. Furthermore, $k^+(Y_j) \geq |U|$ for the set U of all endpoints of such edges $\{l, f\}$.*

► **Lemma 18.** $|\text{ALG}| \leq |\text{OPT}| + 5 \cdot k_h$.

Proof. We show $|S_j| \leq 4 \cdot k^+(Y_j)$ for each j , which, in combination with Lemma 15, implies the lemma. Consider an arbitrary S_j and the corresponding set Y_j . Further, let h_{j-1} denote the maximum matching computed and used by the algorithm for vertex cover instance \bar{G}_{j-1} , and let $\bar{h}_{j-1} = \{\{e, e'\} \in h_{j-1} \mid \{e, e'\} \in \bar{E}_j\}$. Finally, let h_j denote the matching that the algorithm uses for vertex cover instance \bar{G}_j . That is, h_j is computed by completing \bar{h}_{j-1} with a standard augmenting path algorithm. As already argued, at least half the elements of S_j are not matched by \bar{h}_{j-1} but are matched by h_j (cf. Observation 14).

We bound the number of such elements by exploiting that h_j is constructed from \bar{h}_{j-1} via a standard augmenting path algorithm. By definition, each iteration of the augmenting path algorithm increases the size of the current matching (starting with \bar{h}_{j-1}) by one and, in doing so, matches two new elements. In total, at most $2 \cdot (|h_j| - |\bar{h}_{j-1}|)$ previously unmatched elements become part of the matching. Thus, $|S_j| \leq 4 \cdot (|h_j| - |\bar{h}_{j-1}|)$.

We show $(|h_j| - |\bar{h}_{j-1}|) \leq k^+(Y_j)$. According to *Kőnig-Egerváry's Theorem* (e.g., [14]), the size of h_j is equal to the size $|VC_j|$ of the minimum vertex cover for \bar{G}_j . We show $|VC_j| \leq |\bar{h}_{j-1}| + k^+(Y_j)$, which implies $(|h_j| - |\bar{h}_{j-1}|) = |VC_j| - |\bar{h}_{j-1}| \leq k^+(Y_j)$, and, thus, the claim. Let $\overline{VC}_{j-1} = \{e \in VC_{j-1} \mid \exists e' \text{ s.t. } \{e, e'\} \in \bar{h}_{j-1}\}$, then $|\overline{VC}_{j-1}| = |\bar{h}_{j-1}|$.

We prove that we can construct a vertex cover for \bar{G}_j by adding at most $k^+(Y_j)$ elements to \overline{VC}_{j-1} , which implies $|VC_j| \leq |\bar{h}_{j-1}| + k^+(Y_j)$. Consider vertex cover instance \bar{G}_j and set \overline{VC}_{j-1} . By definition, \overline{VC}_{j-1} covers all edges that are part of partial matching \bar{h}_{j-1} .

Consider the elements of \bar{V}_j that are an endpoint of an edge in $\{e, f\} \in \bar{E}_j \Delta \bar{E}_{j-1}$ with e, f non-trivial in G_j . By Lemma 17, $k^-(e) \geq 1$ for each such e and $k^+(Y_j) \geq |U|$ for the set U of all such elements. Thus, we can afford to add U to the vertex cover.

Next, consider an edge $\{e, f\} \in \bar{E}_j$ that is not covered by $\bar{V}C_{j-1} \cup U$. Since $\{e, f\}$ is not covered by U , it must hold that $\{e, f\} \in \bar{E}_j \cap \bar{E}_{j-1}$. Thus, $\{e, f\}$ was covered by VC_{j-1} but is not covered by $\bar{V}C_{j-1}$. This implies $\{e, f\} \cap VC_{j-1} \neq \emptyset$ but $\{e, f\} \cap \bar{V}C_{j-1} = \emptyset$. Assume w.l.o.g. that $e \in VC_{j-1}$. Then, there must be an e' such that $\{e, e'\} \in h_{j-1}$ but $\{e, e'\} \notin \bar{h}_{j-1}$. It follows that $\{e, e'\} \notin \bar{E}_j$. As $\{e, f\}$ is not covered by U , the endpoint e' must be trivial in G_j but non-trivial in G_{j-1} . Thus, e' must have been queried (i) as a mandatory element (or a matching partner) in Line 6 or 13, (ii) as part of VC_{j-1} in Line 5 or (iii) in Line 12. Case (ii) implies $e' \in VC_{j-1}$, contradicting $e \in VC_{j-1}$. Cases (i) or (iii) imply a query to the matching partner e of e' , which contradicts $\{e, f\} \in \bar{E}_j$ (as e would be trivial). Thus, $\{e, f\}$ is covered by $\bar{V}C_{j-1} \cup U$, which implies that $\bar{V}C_{j-1} \cup U$ is a vertex cover for \bar{G}_j . Lemma 17 implies $|U| \leq k^+(Y_j)$. So, $|VC_j| \leq |\bar{h}_{j-1}| + k^+(Y_j)$ which concludes the proof. ◀

Lemmas 16 and 18 imply Theorem 13. Combining Theorems 6 and 13, we show Theorem 12.

6 Further research directions

Plenty other (optimization) problems seem natural in the context of explorable uncertainty with untrusted predictions. For our problem, it would be nice to close the gap in the robustness. We expect that our results extend to all matroids as it does in the classical setting. While we ask for the minimum number of queries to solve a problem *exactly*, it is natural to ask for approximate solutions. The bad news is that for the MST problem there is no improvement over the robustness guarantee of 2 possible even when allowing an arbitrarily large approximation of the exact solution [43, Section 10]. However, it remains open whether an improved consistency or an error-dependent competitive ratio are possible.

References

- 1 Mohamed Abdel-Nasser, Karar Mahmoud, Osama A. Omer, Matti Lehtonen, and Domenec Puig. Link quality prediction in wireless community networks using deep recurrent neural networks. *Alexandria Engineering Journal*, 59(5):3531–3543, 2020. doi:10.1016/j.aej.2020.05.037.
- 2 Susanne Albers and Alexander Eckl. Explorable uncertainty in scheduling with non-uniform testing times. In *WAOA*, volume 12806 of *Lecture Notes in Computer Science*, pages 127–142. Springer, 2020. doi:10.1007/978-3-030-80879-2_9.
- 3 Susanne Albers and Alexander Eckl. Scheduling with testing on multiple identical parallel machines. In *WADS*, volume 12808 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 2021. doi:10.1007/978-3-030-83508-8_3.
- 4 Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *ITCS*, volume 151 of *LIPIcs*, pages 52:1–52:15, 2020. doi:10.4230/LIPIcs.ITCS.2020.52.
- 5 Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 345–355. PMLR, 2020. URL: <http://proceedings.mlr.press/v119/antoniadis20a.html>.
- 6 Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In *NeurIPS*, 2020.

- 7 Luciana Arantes, Evmiridis Bampis, Alexander V. Kononov, Manthos Letsios, Giorgio Lucarelli, and Pierre Sens. Scheduling under uncertainty: A query-based approach. In *IJCAI*, pages 4646–4652. ijcai.org, 2018. doi:10.24963/ijcai.2018/646.
- 8 Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and OR queries. In *ESA*, volume 204 of *LIPICs*, pages 7:1–7:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ESA.2021.7.
- 9 Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In *STOC*, pages 1070–1080. ACM, 2021. doi:10.1145/3406325.3451023.
- 10 Evmiridis Bampis, Konstantinos Dogeas, Alexander V. Kononov, Giorgio Lucarelli, and Fanny Pascual. Speed scaling with explorability uncertainty. In *SPAA*, pages 83–93. ACM, 2021. doi:10.1145/3409964.3461812.
- 11 Evmiridis Bampis, Christoph Dürr, Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlotter. Orienting (hyper)graphs under explorability stochastic uncertainty. In *ESA*, volume 204 of *LIPICs*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ESA.2021.10.
- 12 Paul Beame, Sarel Har-Peled, Sivaramkrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. Algorithms*, 16(4):52:1–52:27, 2020.
- 13 Paul Beame, Sarel Har-Peled, Sivaramkrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. Algorithms*, 16(4):52:1–52:27, 2020. doi:10.1145/3404867.
- 14 Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986.
- 15 Richard Bruce, Michael Hoffmann, Danny Krizanc, and Rajeev Raman. Efficient update strategies for geometric computing with uncertainty. *Theory Comput. Syst.*, 38(4):411–423, 2005. doi:10.1007/s00224-004-1180-4.
- 16 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. doi:10.1561/22000000024.
- 17 Steven Chaplick, Magnús M. Halldórsson, Murilo S. de Lima, and Tigran Tonoyan. Query minimization under stochastic uncertainty. *Theor. Comput. Sci.*, 895:75–95, 2021. doi:10.1016/j.tcs.2021.09.032.
- 18 Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. An adversarial model for scheduling with testing. *Algorithmica*, 82(12):3630–3675, 2020. doi:10.1007/s00453-020-00742-2.
- 19 Franziska Eberle, Alexander Lindermayr, Nicole Megow, Lukas Nölke, and Jens Schlotter. Robustification of online graph exploration methods. In *AAAI*, 2022.
- 20 Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlotter. Learning-augmented query policies for minimum spanning tree with uncertainty. *CoRR*, abs/2206.15201, 2022.
- 21 Thomas Erlebach and Michael Hoffmann. Minimum spanning tree verification under uncertainty. In *WG 2014: International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 8747 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2014. doi:10.1007/978-3-319-12340-0_14.
- 22 Thomas Erlebach and Michael Hoffmann. Query-competitive algorithms for computing with uncertainty. *Bull. EATCS*, 116, 2015. doi:10.1016/j.tcs.2015.11.025.
- 23 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Theor. Comput. Sci.*, 613:51–64, 2016.
- 24 Thomas Erlebach, Michael Hoffmann, Danny Krizanc, Matús Mihalák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. In *STACS*, volume 1 of *LIPICs*, pages 277–288. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2008. doi:10.4230/LIPICs.STACS.2008.1358.

- 25 Tomás Feder, Rajeev Motwani, Liadan O’Callaghan, Chris Olston, and Rina Panigrahy. Computing shortest paths with uncertainty. *J. Algorithms*, 62(1):1–18, 2007. doi:10.1016/j.jalgor.2004.07.005.
- 26 Tomás Feder, Rajeev Motwani, Rina Panigrahy, Chris Olston, and Jennifer Widom. Computing the median with uncertainty. *SIAM J. Comput.*, 32(2):538–547, 2003. doi:10.1137/S0097539701395668.
- 27 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.
- 28 John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed Bandit Allocation Indices*. Wiley, 2nd edition, 2011.
- 29 Marc Goerigk, Manoj Gupta, Jonas Ide, Anita Schöbel, and Sandeep Sen. The robust knapsack problem with queries. *Comput. Oper. Res.*, 55:12–22, 2015. doi:10.1016/j.cor.2014.09.010.
- 30 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.
- 31 Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2319–2327. PMLR, 2019.
- 32 Anupam Gupta, Haotian Jiang, Ziv Scully, and Sahil Singla. The markovian price of information. In *IPCO*, volume 11480 of *Lecture Notes in Computer Science*, pages 233–246. Springer, 2019. doi:10.1007/978-3-030-17953-3_18.
- 33 Manoj Gupta, Yogish Sabharwal, and Sandeep Sen. The update complexity of selection and related problems. *Theory Comput. Syst.*, 59(1):112–132, 2016. doi:10.1007/s00224-015-9664-y.
- 34 Magnús M. Halldórsson and Murilo Santos de Lima. Query-competitive sorting with uncertainty. *Theor. Comput. Sci.*, 867:50–67, 2022. doi:10.1016/j.tcs.2021.03.021.
- 35 Simon Kahan. A model for data in motion. In *STOC*, pages 267–277. ACM, 1991. doi:10.1145/103418.103449.
- 36 Sanjeev Khanna and Wang Chiew Tan. On computing functions with uncertainty. In *PODS*, pages 171–182. ACM, 2001. doi:10.1145/375551.375577.
- 37 Ravi Kumar, Manish Purohit, Aaron Schild, Zoya Svitkina, and Erik Vee. Semi-online bipartite matching. In *ITCS*, volume 124 of *LIPICs*, pages 50:1–50:20. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPICs.ITCS.2019.50.
- 38 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *SODA*, pages 1859–1877. SIAM, 2020. doi:10.1137/1.9781611975994.114.
- 39 Alexander Lindermayr, Nicole Megow, and Bertrand Simon. Double Coverage with Machine-Learned Advice. In *ITCS*, volume 215 of *LIPICs*, pages 99:1–99:18, 2022. doi:10.4230/LIPICs.ITCS.2022.99.
- 40 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1–24:25, 2021. doi:10.1145/3447579.
- 41 Takanori Maehara and Yutaro Yamaguchi. Stochastic packing integer programs with few queries. *Math. Program.*, 182(1):141–174, 2020. doi:10.1007/s10107-019-01388-x.
- 42 Hanna Mazzawi. Optimally reconstructing weighted graphs using queries. In *SODA*, pages 608–615. SIAM, 2010. doi:10.1137/1.9781611973075.51.
- 43 Nicole Megow, Julie Meißner, and Martin Skutella. Randomization helps computing a minimum spanning tree under uncertainty. *SIAM J. Comput.*, 46(4):1217–1240, 2017. doi:10.1137/16M1088375.
- 44 Arturo I. Merino and José A. Soto. The minimum cost query problem on matroids with uncertainty areas. In *ICALP*, volume 132 of *LIPICs*, pages 83:1–83:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.83.

- 45 Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *ITCS*, volume 151 of *LIPICs*, pages 14:1–14:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ITCS.2020.14.
- 46 Noam Nisan. The demand query model for bipartite matching. In *SODA*, pages 592–599. SIAM, 2021. doi:10.1137/1.9781611976465.36.
- 47 Chris Olston and Jennifer Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB*, pages 144–155. Morgan Kaufmann, 2000.
- 48 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *NeurIPS*, pages 9684–9693, 2018.
- 49 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *SODA*, pages 1834–1845. SIAM, 2020. doi:10.1137/1.9781611975994.112.
- 50 Aviad Rubinfeld, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *ITCS*, volume 94 of *LIPICs*, pages 39:1–39:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ITCS.2018.39.
- 51 Sahil Singla. The price of information in combinatorial optimization. In *SODA*, pages 2523–2532. SIAM, 2018. doi:10.1137/1.9781611975031.161.
- 52 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. doi:10.2307/2332286.
- 53 Alexander Wei. Better and simpler learning-augmented online caching. In *APPROX-RANDOM*, volume 176 of *LIPICs*, pages 60:1–60:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.APPROX/RANDOM.2020.60.
- 54 Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *NeurIPS*, 2020.
- 55 Martin Weitzman. Optimal search for the best alternative. *Econometrica*, 47(3):641–54, 1979. doi:10.2307/1910412.

Faster Exponential-Time Approximation Algorithms Using Approximate Monotone Local Search

Bariş Can Esmér ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, Saarland Informatics Campus, Germany

Ariel Kulik ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Dániel Marx ✉ 

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Daniel Neuen ✉ 

School of Computing Science, Simon Fraser University, Burnaby, Canada

Roohani Sharma ✉ 

Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We generalize the monotone local search approach of Fomin, Gaspers, Lokshtanov and Saurabh [J.ACM 2019], by establishing a connection between parameterized approximation and exponential-time approximation algorithms for *monotone subset minimization* problems. In a *monotone subset minimization* problem the input implicitly describes a non-empty set family over a universe of size n which is closed under taking supersets. The task is to find a minimum cardinality set in this family. Broadly speaking, we use *approximate monotone local search* to show that a parameterized α -approximation algorithm that runs in $c^k \cdot n^{\mathcal{O}(1)}$ time, where k is the solution size, can be used to derive an α -approximation randomized algorithm that runs in $d^n \cdot n^{\mathcal{O}(1)}$ time, where d is the unique value in $(1, 1 + \frac{c-1}{\alpha})$ such that $\mathcal{D}(\frac{1}{\alpha} \parallel \frac{d-1}{c-1}) = \frac{\ln c}{\alpha}$ and $\mathcal{D}(a \parallel b)$ is the Kullback-Leibler divergence. This running time matches that of Fomin et al. for $\alpha = 1$, and is strictly better when $\alpha > 1$, for any $c > 1$. Furthermore, we also show that this result can be derandomized at the expense of a sub-exponential multiplicative factor in the running time.

We use an approximate variant of the exhaustive search as a benchmark for our algorithm. We show that the classic $2^n \cdot n^{\mathcal{O}(1)}$ exhaustive search can be adapted to an α -approximate exhaustive search that runs in time $(1 + \exp(-\alpha \cdot \mathcal{H}(\frac{1}{\alpha})))^n \cdot n^{\mathcal{O}(1)}$, where \mathcal{H} is the entropy function. Furthermore, we provide a lower bound stating that the running time of this α -approximate exhaustive search is the best achievable running time in an oracle model. When compared to approximate exhaustive search, and to other techniques, the running times obtained by approximate monotone local search are strictly better for any $\alpha \geq 1$, $c > 1$.

We demonstrate the potential of approximate monotone local search by deriving new and faster exponential approximation algorithms for VERTEX COVER, 3-HITTING SET, DIRECTED FEEDBACK VERTEX SET, DIRECTED SUBSET FEEDBACK VERTEX SET, DIRECTED ODD CYCLE TRANSVERSAL and UNDIRECTED MULTICUT. For instance, we get a 1.1-approximation algorithm for VERTEX COVER with running time $1.114^n \cdot n^{\mathcal{O}(1)}$, improving upon the previously best known 1.1-approximation running in time $1.127^n \cdot n^{\mathcal{O}(1)}$ by Bourgeois et al. [DAM 2011].

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis; Mathematics of computing \rightarrow Approximation algorithms; Mathematics of computing \rightarrow Probabilistic algorithms

Keywords and phrases parameterized approximations, exponential approximations, monotone local search

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.50

Related Version *Full Version:* <https://arxiv.org/abs/2206.13481>

Funding *Dániel Marx:* Research supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.



© Bariş Can Esmér, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 50; pp. 50:1–50:19

Leibniz International Proceedings in Informatics



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

1 Introduction

A lot of interesting problems are computationally hard as they do not admit polynomial-time algorithms. Still, many of them can be solved significantly faster than exhaustive search. The area of exact exponential algorithms studies the design of such techniques. Typically, for *subset problems*, where the goal is to find a subset of a given n -sized universe U that satisfies some property Π , a solution can be found by enumerating all 2^n subsets of U . Therefore, the goal is to design algorithms that beat this exhaustive search and run in time $\mathcal{O}^*(c^n)$ ¹ for as small $1 < c < 2$ as possible.

Exact Monotone Local Search. In a seminal work Fomin, Gaspers, Lokshtanov and Saurabh [13] showed that one can derive faster exact exponential algorithms for subset problems using a parameterized extension algorithm for the problem at hand. A parameterized extension algorithm for a subset problem additionally takes as input a parameter k and a set $X \subseteq U$, runs in time $\mathcal{O}^*(f(k))$, and outputs a set $S \subseteq U$ of size at most k such that $S \cup X$ is a solution, if such a set exists. Fomin et al. [13, Theorem 1.1] showed that if a subset problem admits a parameterized extension algorithm that runs in time $\mathcal{O}^*(c^k)$ for some absolute constant $c > 1$, then it admits a randomized exact exponential algorithm that runs in time $\mathcal{O}^*((\text{emls}(c))^n)$, where $\text{emls}(c) = 2 - \frac{1}{c}$. Their algorithm, called Exact Monotone Local Search (**Exact-MLS**), is simple and is based on monotone local search: it samples a set X of t elements at random, and then extends the set X to an optimum solution using the parameterized extension algorithm. The non-trivial part of the proof of [13, Theorem 1.1] is to analyze the value of t that optimizes the running time. This simple algorithm outperforms the exhaustive search for all subset problems that have parameterized extension algorithms running in $\mathcal{O}^*(c^k)$ time. Moreover, given the existence of a large number of problems that admit the desired parameterized extension algorithm, it yields the state-of-the-art exact exponential algorithms for several problems [13, Table 1].

Exponential Approximation. Another important algorithmic paradigm which deals with NP-hardness is the design of *approximation* algorithms, which are typically polynomial-time algorithms that compute a solution which is not necessarily optimum but has a worst-case guarantee on its *quality*. Though several NP-hard problems admit such algorithms with constant approximation ratios [30], there are many that do not, under reasonable complexity assumptions, for example DIRECTED FEEDBACK VERTEX SET [28]. Also, there are many for which the approximation guarantees cannot be improved beyond a fixed constant. For example, VERTEX COVER admits a 2-approximation but no $(2 - \epsilon)$ -approximation under the Unique Games Conjecture [17].

For problems where some hardness of approximation has been established, a natural question is to determine the smallest c such that the barriers of this hardness can be broken by taking $\mathcal{O}^*(c^n)$ time. Such algorithms are called *exponential approximation* algorithms and this topic has received attention in, e.g., [1, 2, 3, 7, 25].

Consider *subset minimization* problems where the goal is to find a subset of the n -sized universe U of *minimum cardinality*, also called an *optimum* solution, that satisfies some additional property Π . For any approximation ratio $\alpha \geq 1$, we say that a subset $S \subseteq U$ satisfying the property Π is an α -*approximate solution* if $|S| \leq \alpha \cdot |\text{OPT}|$, where $\text{OPT} \subseteq U$ is an optimum solution. An *exponential α -approximation algorithm* for a subset minimization problem returns an α -approximate solution and runs in $\mathcal{O}^*(c^n)$ time for some $1 < c < 2$.

¹ The \mathcal{O}^* notation hides polynomial factors in the input.

Parameterized Approximation. A *parameterized α -approximation* algorithm for a subset minimization problem additionally takes as input the parameter k , runs in time $\mathcal{O}^*(f(k))$, and outputs a solution of size at most $\alpha \cdot k$, if there exists a solution of size at most k . Analogous to parameterized algorithms, one can define the notion of extension algorithms here (see Section 2). The design of parameterized α -approximation algorithms has been an active area of research in the last few years, yielding a plethora of results for problems that exhibit some hardness either in the parameterized setting or in the approximation setting [4, 5, 8, 10, 11, 12, 14, 15, 16, 18, 21, 22, 23, 24, 26, 27, 29].

Approximate Monotone Local Search (Approximate-MLS). In this paper, we show that one can extend the idea of **Exact-MLS** [13] to *derive faster exponential approximation algorithms from parameterized approximation algorithms*. Let $\mathbf{amls}(\alpha, c)$ be the unique value in $(1, 1 + \frac{c-1}{\alpha})$ such that $\mathcal{D}\left(\frac{1}{\alpha} \parallel \frac{\mathbf{amls}(\alpha, c) - 1}{c - 1}\right) = \frac{\ln c}{\alpha}$ where $\mathcal{D}(a \parallel b)$ is the Kullback-Leibler divergence defined as $\mathcal{D}(a \parallel b) = a \ln\left(\frac{a}{b}\right) + (1 - a) \ln\left(\frac{1 - a}{1 - b}\right)$ (see, e.g., [6]). Our main result can be informally stated as follows.

If a monotone subset minimization problem admits a parameterized extension α -approximation algorithm that runs in $\mathcal{O}^*(c^k)$, then one can derive a randomized α -approximation algorithm that runs in time $\mathcal{O}^*((\mathbf{amls}(\alpha, c))^n)$ (see Theorem 2.1).

Since $\mathbf{amls}(1, c) = \mathbf{emls}(c) = 2 - \frac{1}{c}$ for every $c > 1$, our running time matches that of **Exact-MLS** when $\alpha = 1$.

Recall the **Exact-MLS** algorithm described earlier. The non-trivial part of the proof of [13, Theorem 1.1] is the analysis of the probability that the sampled set is *contained* in an optimum solution. To obtain our result, we use the same algorithm and show that if we allow the sampled set to also contain *some* items from outside the optimum solution, then one can speed-up the resulting exponential α -approximation algorithm. Since our analysis need to take into account the calculations for error in the sampled set, this makes analyzing the choice of t more difficult.

In order to better appreciate the running time of our algorithm, that does not seem to have a closed-form formula, we give a mathematical comparison of $\mathbf{amls}(\alpha, c)$ with various benchmark exponential approximation algorithms, showing that our algorithm outperforms all of them.

Benchmark 1: Brute-Force for Exponential Approximation. Since exhaustive search is a trivial benchmark against which the running times of (exact) exponential algorithms are measured, an important question to address is: *how much time does exhaustive search take to find an α -approximate solution?*

Consider a subset minimization problem that is also monotone, that is, for every $S \subseteq T \subseteq U$, if S is a solution, then T is also a solution. We show that for monotone subset minimization problems, the classic brute-force approach can be generalized to an *α -approximation brute-force algorithm* running in time $\mathcal{O}^*(\mathbf{brute}(\alpha)^n)$, where $\mathbf{brute}(\alpha) = 1 + \frac{(\alpha-1)^{\alpha-1}}{\alpha^\alpha} = 1 + \exp\left(-\alpha \cdot \mathcal{H}\left(\frac{1}{\alpha}\right)\right)$ and $\mathcal{H}(\alpha) = -\alpha \ln \alpha - (1 - \alpha) \ln(1 - \alpha)$ denotes the entropy function². The running time essentially follows by showing that a uniformly sampled set of $\alpha \cdot |\text{OPT}|$ items is an α -approximate solution with probability at least $\mathbf{brute}(\alpha)^{-n}$ (up to polynomial factors,

² We adopt the convention that $0^0 = 1$.

see Theorem 5.1). We complement this result by showing that this running time is *best possible*, given only membership oracle access to the problem (Theorem 5.1). For example, for $\alpha = 2$, this brute-force algorithm runs in time $\mathcal{O}^*(1.25^n)$. For $\alpha = 1.1$, it runs in time $\mathcal{O}^*(1.716^n)$ (see also Table 1).

Benchmark 2: Naive conversion from parameterized approximation to exponential approximation. Yet another upper bound on the running time of exponential approximation algorithms can be given as follows. Suppose there is a parameterized α -approximation for a monotone subset minimization problem that runs in $\mathcal{O}^*(c^k)$ time. Run the parameterized algorithm for every value of the parameter k between 0 to $\frac{n}{\alpha}$, and return the solution of minimum cardinality among the solutions returned by the parameterized algorithm. If no solution was found by the parameterized algorithm, then return the whole universe. It can be easily verified that this indeed yields an α -approximation algorithm with running time $\mathcal{O}^*(\text{naive}(\alpha, c)^n)$, where $\text{naive}(\alpha, c) = c^{\frac{1}{\alpha}}$. Observe that for large values of c and appropriate α , $\text{naive}(\alpha, c)$ could be much larger than even 2. But for smaller values of c , it could sometimes beat the brute-force approximation (see Section 2.4).

Benchmark 3: Exact-MLS in the approximate setting. From the description of the Exact-MLS algorithm, it is not difficult to deduce that given a parameterized extension α -approximation for a monotone subset minimization problem that runs in time $\mathcal{O}^*(c^k)$, one can derive an exponential α -approximation for the problem that runs in time $\mathcal{O}^*(\text{eMLS}(c)^n)$. This trivial generalization of Exact-MLS to the approximate setting already performs better than the naive conversion in cases when c is small. For example, VERTEX COVER has a parameterized (extension) 1.1-approximation algorithm that runs in time $\mathcal{O}^*(1.1652^k)$ [3]. Exact-MLS gives a 1.1-approximation running in time $\mathcal{O}^*(1.1417^n)$ whereas the naive conversion gives a running time of $\mathcal{O}^*(1.1462^n)$.

Comparisons. As stated earlier, we show that (see Lemma 1.1) Approximate-MLS is *strictly* faster than the brute-force algorithm (Benchmark 1) or the naive-conversion approach (Benchmark 2) described earlier, for every $\alpha \geq 1$ and $c > 1$. In fact, Lemma 1.1 shows that $\text{aMLS}(\alpha, c)$ converges to $\text{brute}(\alpha)$ as $c \rightarrow \infty$, which would be the expected behavior, because as the parameterized algorithm becomes “less useful”, the running time of our algorithm gets closer to the running time possible without the use of any problem-specific algorithm. Since $\text{aMLS}(1, c) = \text{eMLS}(c) = 2 - \frac{1}{c}$, Lemma 1.1 also shows that the running time is strictly better than that of Exact-MLS (Benchmark 3) when $\alpha > 1$.

► **Lemma 1.1** (\star^3). *For every $c > 1$ the following holds:*

1. $\text{aMLS}(\alpha, c) < \min\{\text{brute}(\alpha), \text{naive}(\alpha, c)\}$ for every $\alpha \geq 1$. In fact, $\text{aMLS}(\alpha, c) \xrightarrow{c \rightarrow \infty} \text{brute}(\alpha)$.
2. $\text{aMLS}(\alpha, c) < \text{eMLS}(c)$ for every $\alpha > 1$. In fact, $\text{aMLS}(\alpha, c)$ is a strictly decreasing function of α .

Applications and Derandomization. We show in Section 2.4 that Approximate-MLS can be used to derive new and faster exponential approximation algorithms for VERTEX COVER, 3-HITTING SET, DIRECTED FEEDBACK VERTEX SET, DIRECTED SUBSET FEEDBACK VERTEX SET, DIRECTED ODD CYCLE TRANSVERSAL and UNDIRECTED MULTICUT. We also show in Section 4 that, as in [13], our algorithm can be derandomized at the expense of a multiplicative sub-exponential factor in the running time.

³ The proofs of statements marked with \star appear in the full version of the paper [9].

2 Definitions and Our Results

2.1 Formal Definitions

We now give some formal definitions that will be required to formally state and describe our main results. An *implicit set system* is a function Φ that takes as input a string $I \in \{0, 1\}^*$, called an *instance*, and returns a set system (U_I, \mathcal{F}_I) where U_I is a universe and \mathcal{F}_I is a collection of subsets of U_I . We use n to denote the size of the universe, that is, $n := |U_I|$. We say that an implicit set system Φ is *polynomial-time computable* if there are two polynomial-time algorithms, the first one, given $I \in \{0, 1\}^*$, computes the set U_I , and the second one, given $S \subseteq U_I$, correctly decides if $S \in \mathcal{F}_I$. A family $\mathcal{F} \subseteq 2^U$ is called *monotone* if $U \in \mathcal{F}$ and for every $S \subseteq T \subseteq U$, if $S \in \mathcal{F}$, then $T \in \mathcal{F}$. We say that an implicit set system Φ is *monotone* if \mathcal{F}_I is monotone for every input I . Throughout the remainder of this work, we only deal with implicit set systems that are polynomial-time computable and monotone. So for the sake of convenience, we refer to a polynomial-time computable and monotone implicit set system simply as an implicit set system.

For an implicit set system Φ , the problem $\Phi_{\text{MIN-SUBSET}}$ takes as input a string $I \in \{0, 1\}^*$ and asks to find $S \in \mathcal{F}_I$ such that $|S|$ is minimum. We refer to the sets in \mathcal{F}_I as solutions of I and we call the sets in \mathcal{F}_I of minimum cardinality as minimum solutions or optimal solutions of I . For $\alpha \geq 1$, we say that an algorithm is a (randomized) α -approximation algorithm for $\Phi_{\text{MIN-SUBSET}}$ if on input I , it returns a set $S \in \mathcal{F}_I$ such that $|S| \leq \alpha \cdot |\text{OPT}|$ (with a constant probability), where OPT is an optimum solution of $\Phi_{\text{MIN-SUBSET}}$.

One can observe that many fundamental graph problems, such as VERTEX COVER, FEEDBACK VERTEX SET, DIRECTED FEEDBACK VERTEX SET, etc., can be cast as a $\Phi_{\text{MIN-SUBSET}}$ problem. Consider for example the VERTEX COVER problem. Given a graph $G = (V, E)$ we say a subset $S \subseteq V$ is a *vertex cover* if for every $(u, v) \in E$ it holds that $u \in S$ or $v \in S$. The input for the VERTEX COVER problem is a graph G and the objective is to find a vertex cover S of G such that $|S|$ is minimum. We can cast VERTEX COVER as a $\Phi_{\text{MIN-SUBSET}}$ problem for the implicit set system Φ_{VC} defined as follows. The instance of the problem is interpreted as a graph $G = (V, E)$. We define the universe U_G as the set of vertices V and the set of solutions $\mathcal{F}_G = \{S \subseteq V \mid S \text{ is a vertex cover of } G\}$ is the set of all vertex covers of G . Finally, we define $\Phi_{\text{VC}}(G) = (U_G, \mathcal{F}_G)$. It can be easily verified that Φ_{VC} is an implicit set system (i.e., it is polynomial-time computable and monotone).

We say an algorithm is a *parameterized (randomized) α -approximate Φ -extension* if, given an instance $I \in \{0, 1\}^*$, $X \subseteq U_I$ and a parameter $k \in \mathbb{N}$, it returns a set $Y \subseteq U_I$ which satisfies the following property (with a constant probability): if there exists a set $S \subseteq U_I$ such that $S \cup X \in \mathcal{F}_I$ and $|S| \leq k$, then it holds that $Y \cup X \in \mathcal{F}_I$ and $|Y| \leq \alpha \cdot k$. We use the shorthand *(randomized) (α, Φ) -extension algorithm* to refer to a *parameterized (randomized) α -approximate Φ -extension algorithm*. Observe that a parameterized α -approximation algorithm for VERTEX COVER can be turned into an $(\alpha, \Phi_{\text{VC}})$ -extension algorithm with the same running time, by taking the instance (G, X, k) of the $(\alpha, \Phi_{\text{VC}})$ -extension algorithm, and running the parameterized α -approximation algorithm on the instance $(G - X, k)$. Observe that this way of converting parameterized α -approximation algorithms to (α, Φ) -extension algorithms holds for various implicit set systems Φ , for example, when Φ corresponds to a vertex deletion problem to a hereditary graph class.

2.2 Our results

Given an (α, Φ) -extension algorithm with running time $\mathcal{O}^*(c^k)$ we design an α -approximation algorithm for $\Phi_{\text{MIN}}\text{-SUBSET}$ with running time $\mathcal{O}^*(\text{amls}(\alpha, c)^n)$ where amls is defined as the unique value $\gamma \in (1, 1 + \frac{c-1}{\alpha})$ such that $\mathcal{D}\left(\frac{1}{\alpha} \parallel \frac{\gamma-1}{c-1}\right) = \frac{\ln c}{\alpha}$. Note that $\text{amls}(\alpha, c)$ is indeed well-defined because for every $\alpha \geq 1$, the function $f(\delta) := \mathcal{D}\left(\frac{1}{\alpha} \parallel \delta\right)$ is monotonically decreasing in the interval $\delta \in (0, \frac{1}{\alpha})$ as well as $f(\delta) \xrightarrow{\delta \rightarrow 0} \infty$ and $f(\delta) \xrightarrow{\delta \rightarrow \frac{1}{\alpha}} 0$.

► **Theorem 2.1** (Approximate Monotone Local Search). *Let Φ be an implicit set system and $\alpha \geq 1$. If there is a randomized (α, Φ) -extension algorithm that runs in time $\mathcal{O}^*(c^k)$, then there is a randomized α -approximation algorithm for $\Phi_{\text{MIN}}\text{-SUBSET}$ that runs in time $\mathcal{O}^*(\text{amls}(\alpha, c)^n)$.*

The formula for $\text{amls}(\alpha, c)$ (which describes the running time of Theorem 2.1) is not a closed-form formula, and we do not expect a closed-form formula for general α, c to exist. However, it represents a *tight* analysis of our algorithm. Despite being represented as an implicit formula, its basic properties can be deduced (see Lemma 1.1). Also, amls can be easily evaluated for every $\alpha, c > 1$. Indeed, for every $\alpha \geq 1$, the function $f(\gamma) = \mathcal{D}\left(\frac{1}{\alpha} \parallel \frac{\gamma-1}{c-1}\right)$ is monotonically decreasing in the interval $(1, 1 + \frac{c-1}{\alpha})$. This means that $\text{amls}(\alpha, c)$ can be evaluated to an arbitrary precision, for every $\alpha \geq 1$ and $c > 1$, using binary search. In particular, the running time implied by Theorem 2.1 can be evaluated.

Theorem 2.1 can be used to obtain faster (than the state-of-art) exponential approximation algorithms for some $\Phi_{\text{MIN}}\text{-SUBSET}$ problems. For example, the brute-force 1.1-approximation algorithm runs in time $\mathcal{O}^*(\text{brute}(1.1)^n) = \mathcal{O}^*(1.716^n)$. The best parameterized 1.1-approximation for VERTEX COVER runs in time $\mathcal{O}^*(1.1652^k)$ [18], where k is the parameter. Using the naive conversion, we obtain a 1.1-approximation that runs in time $\mathcal{O}^*(\text{naive}(1.1, 1.1652)^n) = \mathcal{O}^*(1.149^n)$. The previously known fastest 1.1-approximation algorithm for VERTEX COVER runs in time $\mathcal{O}^*(1.127^n)$ [3]. Using Theorem 2.1 in conjunction with the $\mathcal{O}^*(1.1652^k)$ algorithm of [18], we get a 1.1-approximation algorithm for VERTEX COVER with running time $\mathcal{O}^*(1.114^n)$, improving over all of the above. We provide additional applications in Section 2.4.

In Section 4, we show that the algorithm of Theorem 2.1 can be derandomized, at the cost of a sub-exponential factor in the running time, by generalizing the construction of set inclusion families from [13].

► **Theorem 2.2** (Derandomization Approximate Monotone Local Search). *Let Φ be an implicit set system and $\alpha \geq 1$. If there is an (α, Φ) -extension algorithm that runs in time $\mathcal{O}^*(c^k)$, then there is an α -approximation algorithm for $\Phi_{\text{MIN}}\text{-SUBSET}$ that runs in time $\mathcal{O}^*\left((\text{amls}(\alpha, c))^{n+o(n)}\right)$.*

2.3 Approximate Monotone Local Search

We now present the algorithm underlying Theorem 2.1 and give a sketch for its analysis. Recall Φ is the implicit set family and $\alpha \geq 1$ is the approximation ratio. Let \mathcal{A}_{ext} denote the (α, Φ) -extension algorithm that runs in time $\mathcal{O}^*(c^k)$ where k is the parameter. Given an instance $I \in \{0, 1\}^*$, let $\Phi(I) = (U_I, \mathcal{F}_I)$. The algorithm for Theorem 2.1 is described in Algorithm 2, and it is denoted by **Approximate-MLS**. It uses the subroutine **Sample** (Algorithm 1) which samples a random set from U_I , which is subsequently extended, using \mathcal{A}_{ext} , to yield the solution. Algorithm 2 coincides with the algorithm of [13, Theorem 1.1] when $\alpha = 1$.

Algorithm 1 $\text{Sample}(I, k, t)$.

Input: $I \in \{0, 1\}^*$, $k \in \mathbb{N}$, $t \in \mathbb{N}$

- 1: Sample a set X of size t from U_I uniformly at random.
 - 2: $Y \leftarrow \mathcal{A}_{\text{ext}}(I, X, k - \lceil \frac{t}{\alpha} \rceil)$.
 - 3: $Z \leftarrow X \cup Y$.
 - 4: If $Z \in \mathcal{F}_I$ and $|Z| \leq \alpha \cdot k$, then **return** Z , otherwise **return** U_I .
-

Let OPT be an optimum solution of the instance I of $\Phi_{\text{MIN}}\text{-SUBSET}$. Consider the execution of Sample on the instance (I, k, t) where $k = |\text{OPT}|$. In Step 1 of Sample if $|X \cap \text{OPT}| \geq \frac{t}{\alpha}$ then $|\text{OPT} \setminus X| \leq k - \frac{t}{\alpha}$. Therefore, in Step 2 \mathcal{A}_{ext} must return a set Y such that $X \cup Y \in \mathcal{F}_I$ and $|Y| \leq \alpha \cdot (k - \frac{t}{\alpha}) = \alpha k - t$. Thus, the set $Z = X \cup Y$ computed in Step 3 is an α -approximate solution of I . Let $\text{hyper}(n, k, t, x)$ be the probability that a uniformly random set X of t items out of $[n] := \{1, \dots, n\}$ satisfies $|X \cap [k]| \geq x$. The distribution of $|X \cap [k]|$ is commonly referred as *hyper-geometric*. Since $\Pr(|X \cap \text{OPT}| \geq \frac{t}{\alpha}) = \text{hyper}(n, k, t, \frac{t}{\alpha})$, Sample returns an α -approximate solution of I with probability $\text{hyper}(n, k, t, \frac{t}{\alpha})$. Observe that the running time of the Sample subroutine is proportional (up to polynomial factors) to the running time of the call to \mathcal{A}_{ext} in Step 2. Thus, the Sample subroutine runs in time $\mathcal{O}^*(c^{k - \frac{t}{\alpha}})$.

Algorithm 2 $\text{Approximate-MLS}(I)$.

Input: $I \in \{0, 1\}^*$

- 1: Define $n = |U_I|$ and $\mathcal{S} \leftarrow \emptyset$.
 - 2: **for** k from 0 to $\frac{n}{\alpha}$ **do**
 - 3: $t \leftarrow \operatorname{argmin}_{t \in [0, \alpha k] \cap \mathbb{N}} \left(\frac{c^{k - \frac{t}{\alpha}}}{\text{hyper}(n, k, t, \frac{t}{\alpha})} \right)$.
 - 4: Run $\mathcal{S} = \mathcal{S} \cup \{\text{Sample}(I, k, t)\}$ for $(\text{hyper}(n, k, t, \frac{t}{\alpha}))^{-1}$ times.
 - 5: **Return** a minimum sized set in \mathcal{S} .
-

Now, let us consider the execution of Approximate-MLS on input I . The analysis of Approximate-MLS focuses on the iteration of Step 2 when $k = |\text{OPT}|$. In this iteration, each call to $\text{Sample}(I, k, t)$ returns an α -approximate solution with probability $\text{hyper}(n, k, t, \frac{t}{\alpha})$ (as argued above). Since in Step 4, Sample is invoked $(\text{hyper}(n, k, t, \frac{t}{\alpha}))^{-1}$ times, at the end of the execution of Step 4, the set \mathcal{S} contains an α -approximate solution of I with a constant probability.

The running time of a *fixed iteration* in Step 2 of Approximate-MLS is $(\text{hyper}(n, k, t, \frac{t}{\alpha}))^{-1}$ times the running time of Sample , that is, $\frac{c^{k - \frac{t}{\alpha}}}{\text{hyper}(n, k, t, \frac{t}{\alpha})}$. Let us denote $\text{iteration}_{n, k, c}(t) = \frac{c^{k - \frac{t}{\alpha}}}{\text{hyper}(n, k, t, \frac{t}{\alpha})}$. Observe that the value of t selected in Step 3 minimizes $\text{iteration}_{n, k, c}(t)$. From the algorithmic perspective the selection of the optimal value of t is straightforward as $\text{iteration}_{n, k, c}(t)$ can be computed in polynomial time for each value of t (given n and k). However, the asymptotic analysis of $\text{iteration}_{n, k, c}(t)$, and hence the overall running time, requires an in-depth understanding of the random process and serves as the main technical contribution of this paper.

As described earlier, when $\alpha = 1$, Algorithm 2 coincides with the algorithm of [13, Theorem 1.1]. The analysis of [13, Theorem 1.1] lower bounds the probability that the set X sampled on Step 1 of Sample satisfies $X \subseteq \text{OPT}$. The analysis of our algorithm lower bounds the probability that $\frac{|X \cap \text{OPT}|}{|X|} \geq \frac{1}{\alpha}$. In particular, the sampling step may select items which

are not in OPT , though the number of such items is restricted. This allows for an improved running time in comparison to that of [13] (see Lemma 1.1), but renders the analysis of the running time to be more involved.

For the analytical estimation of t , which is selected in Step 3 of **Approximate-MLS**, the question that one needs to understand is *how many items should the algorithm sample before it decides to use \mathcal{A}_{ext} to extend the sampled set*. Assume that the algorithm already sampled a set X of t items such that $|X \cap \text{OPT}| \approx \frac{t}{\alpha}$. Let $\varepsilon > 0$ be some small number. Observe that $U_I \setminus X$ contains $\approx k - \frac{t}{\alpha}$ items from OPT , and thus $\frac{|(U_I \setminus X) \cap \text{OPT}|}{|U_I \setminus X|} \approx \frac{k - \frac{t}{\alpha}}{n - t}$. The algorithm now has two options: it can either further sample a set A of additional $\varepsilon \cdot n$ items or, use \mathcal{A}_{ext} with the parameter $k - \frac{t}{\alpha}$ to extend X to a final solution. In the first case, the time taken to sample a set A of $\varepsilon \cdot n$ items such that $|A \cap \text{OPT}| \geq \frac{|A|}{\alpha} = \frac{\varepsilon \cdot n}{\alpha}$ holds with constant probability, is $(\Pr(|A \cap \text{OPT}| \geq \frac{\varepsilon \cdot n}{\alpha}))^{-1}$. In the second case, the algorithm spends an additional factor of $c^{\frac{\varepsilon \cdot n}{\alpha}}$ time to extend the set X , instead of $X \cup A$, to the final solution. Thus, if $\Pr(|A \cap \text{OPT}| \geq \frac{\varepsilon \cdot n}{\alpha}) > c^{-\frac{\varepsilon \cdot n}{\alpha}}$, it is better to continue sampling, and otherwise, it is better to run \mathcal{A}_{ext} on the instance $(I, X, k - \frac{t}{\alpha})$. Therefore, to understand the analytics of the chosen t , one needs to upper bound $\Pr(|A \cap \text{OPT}| \geq \frac{\varepsilon \cdot n}{\alpha})$.

We view the sampling of A as an iterative process in which the items are sampled one after the other. When sampling the ℓ -th item, the ratio between the remaining items in OPT and the available items is $\approx \frac{k - \frac{t}{\alpha} - \Delta}{n - t - \ell}$, where Δ is the number of items from OPT sampled in previous iterations and $0 \leq \Delta \leq \ell \leq \varepsilon \cdot n$. As $\varepsilon \cdot n$ is small, we estimate $\frac{k - \frac{t}{\alpha} - \Delta}{n - t - \ell} \approx \frac{k - \frac{t}{\alpha}}{n - t}$. Therefore, the probability that the ℓ -th sampled item is in OPT is roughly $\frac{k - \frac{t}{\alpha}}{n - t}$. Thus, $|A \cap \text{OPT}|$ can be estimated as the sum of $\varepsilon \cdot n$ Bernoulli random variables $x_1, \dots, x_{\varepsilon \cdot n}$ with probability $\frac{k - \frac{t}{\alpha}}{n - t}$. Thus,

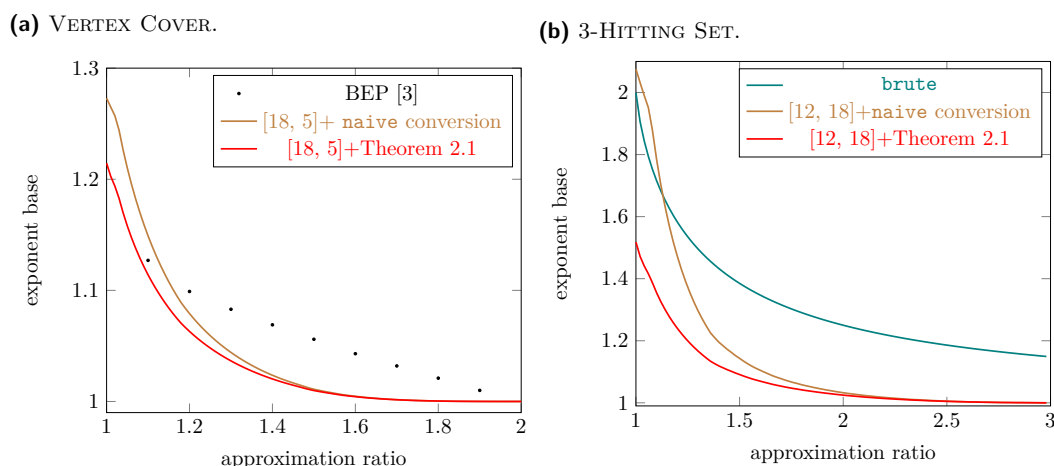
$$\Pr\left(|A \cap \text{OPT}| \geq \frac{\varepsilon \cdot n}{\alpha}\right) \approx \Pr\left(\sum_{i=1}^{\varepsilon \cdot n} x_i \geq \frac{\varepsilon \cdot n}{\alpha}\right) \approx \exp\left(-\varepsilon \cdot n \cdot \mathcal{D}\left(\frac{1}{\alpha} \left\| \frac{k - \frac{t}{\alpha}}{n - t} \right.\right)\right),$$

where the last estimation follows from a large deviation property of binomial distributions [6, Theorem 11.1.4] and assumes $|\text{OPT}| \leq \frac{n}{\alpha}$.

Therefore, the (optimal) selection of t which minimizes $\text{iteration}_{n,k,c}(t)$ is the largest t which satisfies $\exp\left(-\varepsilon \cdot n \cdot \mathcal{D}\left(\frac{1}{\alpha} \left\| \frac{k - \frac{t}{\alpha}}{n - t} \right.\right)\right) > c^{-\frac{\varepsilon \cdot n}{\alpha}}$, or equivalently, $\mathcal{D}\left(\frac{1}{\alpha} \left\| \frac{k - \frac{t}{\alpha}}{n - t} \right.\right) \approx \frac{\ln c}{\alpha}$. We use this value of t to bound the running time of an iteration of Step 2, that is, to upper bound $\min_{t \in [0, \alpha k] \cap \mathbb{N}} \text{iteration}_{n,k,c}(t)$. This analytical estimation of t forms the crux in analyzing the overall running time of Algorithm 2.

2.4 Applications of Approximate-MLS

In this section we use Theorem 2.1 to get faster randomized exponential approximation algorithms for VERTEX COVER, 3-HITTING SET, DIRECTED FEEDBACK VERTEX SET (DFVS), DIRECTED SUBSET FEEDBACK VERTEX SET (SUBSET DFVS), DIRECTED ODD CYCLE TRANSVERSAL (DOCT) and UNDIRECTED MULTICUT. One can observe that all these problems can be described as some Φ_{MIN} -SUBSET problem. Since all these problems can be interpreted as vertex deletion problems to some hereditary graph class, any parameterized α -approximation algorithm for these problems can be used as an (α, Φ) -extension algorithm, for the respective Φ .



■ **Figure 1** Results for VERTEX COVER and 3-HITTING SET. A dot at (α, d) means that the respective algorithm outputs an α -approximation in time $\mathcal{O}^*(d^n)$.

VERTEX COVER (VC). In [3] Bourgeois, Escoffier and Paschos designed several exponential approximation algorithms for VC for approximation ratios in the range $(1, 2)$. For any $\alpha \in (1, 2)$ the best known running time of a parameterized randomized α -approximation algorithm for VC is attained in [18] if $\alpha \gtrsim 1.03$, and in [5] if $\alpha \lesssim 1.03$. We use these algorithms in conjunction with Theorem 2.1 to obtain faster randomized exponential α -approximation algorithms for VC for values of α in the range $(1, 2)$. We compare our running times to the running times obtained by the naive conversion (Benchmark 2) and to the running times in [3].⁴ We present the running time for selected approximation ratios in Table 1 and give a graphical comparison in Figure 1a.

■ **Table 1** Results for VERTEX COVER and 3-HITTING SET. A value d at the column of an approximation ratio α means that the respective algorithm outputs an α -approximation in time $\mathcal{O}^*(d^n)$.

VERTEX COVER

ratio	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
brute(α)	1.716	1.583	1.496	1.433	1.385	1.347	1.317	1.291	1.269
BEP [3]	1.127	1.099	1.083	1.069	1.056	1.043	1.032	1.021	1.01
[18]+Naive Conv.	1.149	1.079	1.044	1.0236	1.0110	1.00469	1.00162	1.000406	1.0000432
[18]+Theorem 2.1	1.114	1.064	1.036	1.0203	1.0099	1.00435	1.00156	1.000397	1.0000428

3-HITTING SET

ratio	1.2	1.4	1.6	1.8	2.0	2.2	2.4	2.6	2.8
brute(α)	1.583	1.433	1.347	1.291	1.251	1.220	1.196	1.177	1.162
[18]+Naive Conv.	1.471	1.196	1.105	1.0582	1.0326	1.0173	1.00831	1.00324	1.000903
[18]+Theorem 2.1	1.240	1.119	1.0698	1.0417	1.0248	1.0140	1.00711	1.00292	1.000853

⁴ The result of [3] provides an α -approximation algorithm for every $\alpha \in (1, 2)$. As the evaluation of these running times is not trivial, we only provide the running times which were explicitly given in [3] for selected approximation ratios.

3-HITTING SET (3-HS). The problem admits a simple polynomial-time 3-approximation algorithm which cannot be improved assuming UGC [17]. For any $\alpha \in (1, 3)$ the best known running time of a parameterized α -approximation algorithm for 3-HS is attained by either [12] if $\alpha \lesssim 1.08$, or [18] if $\alpha \gtrsim 1.08$. Using these algorithms as parameterized extension algorithms, we calculate the running times of α -approximation algorithms for 3-HS attained using the naive conversion (Benchmark 2) and Theorem 2.1, for values of $\alpha \in (1, 3)$. We provide the running times for selected approximation ratios in Table 1 and a graphical comparison in Figure 1b.

DFVS, SUBSET DFVS, DOCT, UNDIRECTED MULTICUT. For all these problems [22] gave parameterized 2-approximation algorithms that run in time $\mathcal{O}^*(c^k)$, for some constant $c > 1$. One can easily observe from the description of the DFVS algorithm in [22] that it runs in time $\mathcal{O}^*(1024^k)$. Using Theorem 2.1 we get that DFVS admits an exponential 2-approximation algorithm that runs in time $\mathcal{O}^*(1.2498^n)$. This running time is significantly better than the running time derived using the naive conversion (Benchmark 2) of the algorithm of [22], which does not give anything meaningful for this problem. It is also significantly better than using **Exact-MLS** with the algorithm of [22], which gives $\mathcal{O}^*((\mathbf{e}\mathbf{m}\mathbf{l}\mathbf{s}(1024))^n) = \mathcal{O}^*(1.9991^n)$. It is also qualitatively better than the brute-force 2-approximation algorithm (Benchmark 1), which runs in time $\mathcal{O}^*(1.25^n)$.

Using Lemma 1.1, we can show that we get faster 2-approximation algorithms for all mentioned problems compared to the brute-force 2-approximation algorithm, or the naive conversion of the parameterized algorithms in [22] or the application of **Exact-MLS** with the algorithms of [22]. Note that even though the algorithms derived from Theorem 2.1 are only *qualitatively* better than brute-force approximation, we emphasize that **Approximate-MLS** is always strictly better than brute-force approximation (and the other benchmarks described earlier). Also, it reflects Part 1 of Lemma 1.1, that as c increases (that is, as the parameterized extension algorithm becomes slower), our algorithm converges to the brute-force approximation.

3 Analysis of Approximate Monotone Local Search

This section is dedicated to the proof of Theorem 2.1. As explained earlier, the algorithm promised in Theorem 2.1 is **Approximate-MLS** (Algorithm 2). In Lemma 3.1 we prove the correctness of Algorithm 2 and in Lemma 3.2 we provide a formula for its running time. Finally, in Lemma 3.4 we upper bound the running time of the formula obtained in Lemma 3.2 with $\mathcal{O}^*(\mathbf{a}\mathbf{m}\mathbf{l}\mathbf{s}(\alpha, c)^n)$, thereby proving Theorem 2.1.

► **Lemma 3.1** (Correctness \star). *Approximate-MLS (Algorithm 2) is a randomized α -approximation algorithm for $\Phi_{\text{MIN-SUBSET}}$.*

► **Lemma 3.2** (Running time). *Approximate-MLS (Algorithm 2) runs in time $f_{\alpha,c}(n) \cdot n^{\mathcal{O}(1)}$ where*

$$f_{\alpha,c}(n) := \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \min_{t \in [0, \alpha k] \cap \mathbb{N}} \frac{c^{k - \frac{t}{\alpha}}}{\mathbf{hyper}(n, k, t, \frac{t}{\alpha})}. \quad (1)$$

Proof. For each choice of k , Algorithm 2 chooses in Step 3 a number t that minimizes $\frac{c^{k - \frac{t}{\alpha}}}{\mathbf{hyper}(n, k, t, \frac{t}{\alpha})}$. Clearly, this step can be performed in time $n^{\mathcal{O}(1)}$. Afterwards, the algorithm calls Algorithm 1 $\mathbf{hyper}(n, k, t, \frac{t}{\alpha})^{-1}$ times which takes $\frac{c^{k - \frac{t}{\alpha}}}{\mathbf{hyper}(n, k, t, \frac{t}{\alpha})} \cdot n^{\mathcal{O}(1)}$ time in total. So overall, the running time is upper bounded by $f_{\alpha,c}(n) \cdot n^{\mathcal{O}(1)}$. ◀

We now proceed with the main part of the analysis which is to bound $f_{\alpha,c}(n)$ by $\text{amls}(\alpha, c)^n$ up to some polynomial factors. We remark at this point (without giving a proof) that our analysis is in fact tight, that is, it can be shown that $f_{\alpha,c}(n)$ is equal to $\text{amls}(\alpha, c)^n$ up to some polynomial factors in n .

Recall that $\mathcal{H}(p) = -p \ln p - (1-p) \ln(1-p)$ denotes the entropy function. We will use the following bound on binomial coefficients (see, e.g., [6, Example 11.1.3]):

$$\frac{1}{n+1} \cdot \exp\left(n \cdot \mathcal{H}\left(\frac{k}{n}\right)\right) \leq \binom{n}{k} \leq \exp\left(n \cdot \mathcal{H}\left(\frac{k}{n}\right)\right) \quad (2)$$

for all $n, k \in \mathbb{N}$ such that $0 \leq k \leq n$.

Moreover, we also need the following technical lemma. Intuitively speaking, it says that small perturbations to the values of a and b do not change the value of $a \cdot \mathcal{H}\left(\frac{b}{a}\right)$ by a large amount.

► **Lemma 3.3** (*). *For $0 \leq b \leq a \leq n$ and $\varepsilon, \delta \in [-1, 1]$ such that $a + \varepsilon \geq 0$ and $0 \leq b + \delta \leq a + \varepsilon$, we have $\left|a \cdot \mathcal{H}\left(\frac{b}{a}\right) - (a + \varepsilon) \cdot \mathcal{H}\left(\frac{b+\delta}{a+\varepsilon}\right)\right| = \mathcal{O}(\log(n))$.*

Finally, recall that $\mathcal{D}(a||b) = a \ln \frac{a}{b} + (1-a) \ln \frac{1-a}{1-b}$ denotes the Kullback-Leibler divergence (see, e.g., [6]).

► **Lemma 3.4.** *It holds that*

$$f_{\alpha,c}(n) = \mathcal{O}^*(\text{amls}(\alpha, c)^n).$$

Proof. Recall that $\text{hyper}(n, k, t, \frac{t}{\alpha})$ denotes the probability that a uniformly random set X of t elements out of $[n]$ satisfies that $|X \cap [k]| \geq \frac{t}{\alpha}$. Thus, we have that

$$\text{hyper}\left(n, k, t, \frac{t}{\alpha}\right) = \sum_{y \geq \lceil \frac{t}{\alpha} \rceil} \frac{\binom{k}{y} \binom{n-k}{t-y}}{\binom{n}{t}} \geq \frac{\binom{k}{\lceil \frac{t}{\alpha} \rceil} \binom{n-k}{t-\lceil \frac{t}{\alpha} \rceil}}{\binom{n}{t}} = \frac{\binom{t}{\lceil \frac{t}{\alpha} \rceil} \binom{n-t}{k-\lceil \frac{t}{\alpha} \rceil}}{\binom{n}{k}}, \quad (3)$$

where the last equality holds since the distribution of $|X \cap [k]|$, where $X \subseteq [n]$ is a uniformly random set of cardinality t , is identical to the distribution of $|Y \cap [t]|$ where $Y \subseteq [n]$ is a uniformly random set of cardinality k .

Using (3) we have

$$\begin{aligned} f_{\alpha,c}(n) &= \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \min_{t \in [0, \alpha k] \cap \mathbb{N}} \frac{c^{k-\frac{t}{\alpha}}}{\text{hyper}(n, k, t, \frac{t}{\alpha})} \leq \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \min_{t \in [0, \alpha k] \cap \mathbb{N}} \frac{c^{k-\frac{t}{\alpha}} \cdot \binom{n}{k}}{\binom{t}{\lceil \frac{t}{\alpha} \rceil} \binom{n-t}{k-\lceil \frac{t}{\alpha} \rceil}} \\ &\leq n^{\mathcal{O}(1)} \cdot \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \binom{n}{k} \exp\left(\min_{t \in [0, \alpha k] \cap \mathbb{N}} \left(\left(k - \frac{t}{\alpha}\right) \ln(c) - t \cdot \mathcal{H}\left(\frac{\lceil \frac{t}{\alpha} \rceil}{t}\right) - (n-t) \cdot \mathcal{H}\left(\frac{k - \lceil \frac{t}{\alpha} \rceil}{n-t}\right) \right)\right) \quad (4) \\ &\leq n^{\mathcal{O}(1)} \cdot \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \binom{n}{k} \exp\left(\min_{t \in [0, \alpha k] \cap \mathbb{N}} \left(\left(k - \frac{t}{\alpha}\right) \ln(c) - t \cdot \mathcal{H}\left(\frac{1}{\alpha}\right) - (n-t) \cdot \mathcal{H}\left(\frac{k - \frac{t}{\alpha}}{n-t}\right) \right)\right) \end{aligned}$$

where the second inequality follows from (2) and the third inequality follows from Lemma 3.3.

Define

$$g_{n,k}(t) := \left(k - \frac{t}{\alpha}\right) \ln(c) - t \cdot \mathcal{H}\left(\frac{1}{\alpha}\right) - (n-t) \cdot \mathcal{H}\left(\frac{k - \frac{t}{\alpha}}{n-t}\right).$$

50:12 Approximate Monotone Local Search

By Lemma 3.3 it holds that $|g_{n,k}(t) - g_{n,k}(t - \varepsilon)| = \mathcal{O}(\log n)$ for any $t \in [0, \alpha k]$ and $0 \leq \varepsilon \leq \min\{1, t\}$. Using this observation and (4) we get

$$\begin{aligned} f_{\alpha,c}(n) &\leq n^{\mathcal{O}(1)} \cdot \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \binom{n}{k} \exp\left(\min_{t \in [0, \alpha k] \cap \mathbb{N}} g_{n,k}(t)\right) \\ &\leq n^{\mathcal{O}(1)} \cdot \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \binom{n}{k} \exp\left(\min_{t \in [0, \alpha k]} g_{n,k}(t)\right). \end{aligned} \quad (5)$$

Observe that in the last term the range of t is not restricted to integral values.

Let $\delta^* = \frac{\text{amls}(\alpha, c) - 1}{c - 1}$. By the definition of amls it holds that $\delta^* \in (0, \frac{1}{\alpha})$ and $\mathcal{D}\left(\frac{1}{\alpha} \parallel \delta^*\right) = \frac{\ln(c)}{\alpha}$. Define $t^*(n, k) := \frac{k - n\delta^*}{\frac{1}{\alpha} - \delta^*}$, thus $\frac{k - \frac{t^*(n, k)}{\alpha}}{n - t^*(n, k)} = \delta^*$ and $\mathcal{D}\left(\frac{1}{\alpha} \parallel \frac{k - \frac{t^*(n, k)}{\alpha}}{n - t^*(n, k)}\right) = \frac{\ln c}{\alpha}$ for every $n \in \mathbb{N}$ and $k \in \mathbb{N}$. It can be verified that $g_{n,k}(t)$ is convex and has a global minimum at $t^*(n, k)$, though this observation is not directly used by our proof.

For any $n \in \mathbb{N}$ and $k \in [0, \frac{n}{\alpha}] \cap \mathbb{N}$ it holds that $t^*(n, k) = \frac{\alpha k (\frac{1}{\alpha} - \delta^*) + \alpha k \delta^* - n \delta^*}{\frac{1}{\alpha} - \delta^*} \leq \alpha k$ since $\alpha k \leq n$. Furthermore, $t^*(n, k) \geq 0$ if and only if $k \geq n\delta^*$. Following this observation we partition the summation in (5) into two parts. Define

$$A(n) = \sum_{k=0}^{\lfloor n\delta^* \rfloor} \binom{n}{k} \exp\left(\min_{t \in [0, \alpha k]} g_{n,k}(t)\right) \text{ and } B(n) = \sum_{k=\lfloor n\delta^* \rfloor + 1}^{\lfloor \frac{n}{\alpha} \rfloor} \binom{n}{k} \exp\left(\min_{t \in [0, \alpha k]} g_{n,k}(t)\right).$$

Thus, $f_{\alpha,c}(n) \leq n^{\mathcal{O}(1)} \cdot (A(n) + B(n))$. We bound each of the sums $A(n)$ and $B(n)$ separately.

In order to bound $B(n)$ we use the following algebraic identity.

▷ **Claim 3.5** (\star). It holds that

$$g_{n,k}(t) = \left(k - \frac{t}{\alpha}\right) \ln(c) + t \cdot \mathcal{D}\left(\frac{1}{\alpha} \parallel \frac{k - \frac{t}{\alpha}}{n - t}\right) + k \cdot \ln\left(\frac{k - \frac{t}{\alpha}}{n - t}\right) + (n - k) \cdot \ln\left(1 - \frac{k - \frac{t}{\alpha}}{n - t}\right).$$

For any $n \in \mathbb{N}$ and $k \in [n\delta^*, \frac{n}{\alpha}] \cap \mathbb{N}$ it holds that $0 \leq t^*(n, k) \leq \alpha k$. Thus,

$$\begin{aligned} \min_{t \in [0, \alpha k]} g_{n,k}(t) &\leq g_{n,k}(t^*(n, k)) \\ &= \left(k - \frac{t^*(n, k)}{\alpha}\right) \ln(c) + t^*(n, k) \cdot \mathcal{D}\left(\frac{1}{\alpha} \parallel \delta^*\right) + k \cdot \ln(\delta^*) + (n - k) \cdot \ln(1 - \delta^*) \\ &= k \cdot \ln(c) + k \cdot \ln(\delta^*) + (n - k) \cdot \ln(1 - \delta^*) \\ &= k \cdot \ln\left(\frac{c \cdot \delta^*}{1 - \delta^*}\right) + n \cdot \ln(1 - \delta^*), \end{aligned}$$

where the first equality uses Claim 3.5 and the second equality follows from $\mathcal{D}\left(\frac{1}{\alpha} \parallel \delta^*\right) = \frac{\ln(c)}{\alpha}$. Therefore,

$$\begin{aligned} B(n) &= \sum_{k=\lfloor n\delta^* \rfloor + 1}^{\lfloor \frac{n}{\alpha} \rfloor} \binom{n}{k} \cdot \exp\left(\min_{t \in [0, \alpha k]} g_{n,k}(t)\right) \leq \sum_{k=0}^n \binom{n}{k} \left(\frac{c \cdot \delta^*}{1 - \delta^*}\right)^k (1 - \delta^*)^n \\ &= (1 - \delta^*)^n \left(\frac{c \cdot \delta^*}{1 - \delta^*} + 1\right)^n = ((c - 1)\delta^* + 1)^n \end{aligned} \quad (6)$$

using the Binomial Theorem.

We now proceed to bound $A(n)$. For every $n \in \mathbb{N}$ and $0 \leq k \leq \delta^* n$ it holds that

$$\min_{t \in [0, \alpha k]} g_{n,k}(t) \leq g_{n,k}(0) = k \cdot \ln(c) - n \cdot \mathcal{H}\left(\frac{k}{n}\right) \leq k \cdot \ln c - \ln \binom{n}{k},$$

where the last inequality follows from (2). Therefore,

$$A(n) = \sum_{k=0}^{\lfloor n\delta^* \rfloor} \binom{n}{k} \cdot \exp\left(\min_{t \in [0, \alpha k]} g_{n,k}(t)\right) \leq \sum_{k=0}^{\lfloor n\delta^* \rfloor} c^k \leq n \cdot (c^{\delta^*})^n. \quad (7)$$

Finally, by using (6) and (7), we get

$$\begin{aligned} f_{\alpha,c}(n) &\leq n^{\mathcal{O}(1)} \cdot (A(n) + B(n)) \leq n^{\mathcal{O}(1)} \cdot \left((c^{\delta^*})^n + ((c-1)\delta^* + 1)^n\right) \\ &\leq n^{\mathcal{O}(1)} \cdot ((c-1)\delta^* + 1)^n, \end{aligned}$$

where the third inequality uses $c^{\delta^*} \leq (c-1)\delta^* + 1$ which holds because $f(x) := c^x - (c-1)x - 1$ is convex and has two roots at 0 and 1. By the definition of δ^* it holds that $(c-1)\delta^* + 1 = \text{aml}_s(\alpha, c)$ and thus, $f_{\alpha,c}(n) \leq n^{\mathcal{O}(1)} \cdot \text{aml}_s(\alpha, c)^n$. \blacktriangleleft

Finally, Theorem 2.1 is implied by Lemmas 3.1, 3.2 and 3.4.

4 Derandomization

In this section, we show how to derandomize Algorithm 2. In particular, we prove Theorem 2.2. As usual, let (U_I, \mathcal{F}_I) be a set system and let $k, t, n \in \mathbb{N}$ be the variables from Algorithm 2 and let $\alpha \geq 1$. In order to derandomize the algorithm, it is sufficient to find a collection \mathcal{C} of subsets of U_I of size t such that, for every possible solution set $S \subseteq U_I$ of size k , there is some set $X \in \mathcal{C}$ such that $|X \cap S| \geq \frac{t}{\alpha}$. We refer to such a family \mathcal{C} as an $(n, k, t, \frac{t}{\alpha})$ -set-intersection-family which is formally defined below.

► **Definition 4.1.** *Let U be a universe of size n and let $p, q, r \geq 1$ such that $n \geq p \geq r$ and $n - p + r \geq q \geq r$. A family $\mathcal{C} \subseteq \binom{U}{q}$ is a (n, p, q, r) -set-intersection-family if for every $T \in \binom{U}{p}$ there is some $X \in \mathcal{C}$ such that $|T \cap X| \geq r$.*

Given a $(n, k, t, \frac{t}{\alpha})$ -set-intersection-family \mathcal{C} we can derandomize Algorithm 2 by iterating over all choices $X \in \mathcal{C}$ instead of repeatedly sampling a set X uniformly at random. Observe that the derandomized algorithm (for a fixed k, t) runs in time $\mathcal{O}^*(|\mathcal{C}| \cdot c^{k - \frac{t}{\alpha}})$. Now, we define

$$\kappa(n, p, q, r) := \frac{\binom{n}{q}}{\binom{p}{r} \cdot \binom{n-p}{q-r}}.$$

The following theorem computes the desired set-intersection-family of small size.

► **Theorem 4.2** (\star). *There is an algorithm that, given a set U of size n and $p, q, r \geq 1$ such that $n \geq p \geq r$ and $n - p + r \geq q \geq r$, computes an (n, p, q, r) -set-intersection-family of size $\kappa(n, p, q, r) \cdot 2^{\mathcal{O}(n)}$ in time $\kappa(n, p, q, r) \cdot 2^{\mathcal{O}(n)}$.*

For the proof of Theorem 4.2 we extend the arguments from [13] which provide such a result for the special case when $q = r$ (which corresponds to the case $\alpha = 1$).

50:14 Approximate Monotone Local Search

Proof of Theorem 2.2. We proceed analogously to the proof of Theorem 2.1 with the following changes. In Algorithm 2, Step 3 we define

$$t := \operatorname{argmin}_{t \in [0, \alpha k] \cap \mathbb{N}} \kappa \left(n, k, t, \left\lceil \frac{t}{\alpha} \right\rceil \right) c^{k - \frac{t}{\alpha}}$$

and then compute an $(n, k, t, \frac{t}{\alpha})$ -set-intersection-family \mathcal{C} of size $\kappa(n, p, q, r) \cdot 2^{o(n)}$ using Theorem 4.2. Afterwards, we repeatedly execute Algorithm 1 for every $X \in \mathcal{C}$ (instead of sampling X uniformly at random). Repeating the analysis of Lemma 3.2 the running time is bounded by $\mathcal{O}^*(f_{\alpha, c}(n))$ where,

$$f_{\alpha, c}(n) = \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \min_{t \in [0, \alpha k] \cap \mathbb{N}} \kappa \left(n, k, t, \left\lceil \frac{t}{\alpha} \right\rceil \right) c^{k - \frac{t}{\alpha}} \cdot 2^{o(n)}.$$

As we already proved in Lemma 3.4 it holds that

$$\begin{aligned} \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \min_{t \in [0, \alpha k] \cap \mathbb{N}} \kappa \left(n, k, t, \left\lceil \frac{t}{\alpha} \right\rceil \right) c^{k - \frac{t}{\alpha}} &= \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \min_{t \in [0, \alpha k] \cap \mathbb{N}} \frac{c^{k - \lceil \frac{t}{\alpha} \rceil} \cdot \binom{n}{t}}{\binom{k}{\lceil \frac{t}{\alpha} \rceil} \binom{n-k}{t - \lceil \frac{t}{\alpha} \rceil}} \\ &= \sum_{k=0}^{\lfloor \frac{n}{\alpha} \rfloor} \min_{t \in [0, \alpha k] \cap \mathbb{N}} \frac{c^{k - \lceil \frac{t}{\alpha} \rceil} \cdot \binom{n}{k}}{\binom{t}{\lceil \frac{t}{\alpha} \rceil} \binom{n-t}{k - \lceil \frac{t}{\alpha} \rceil}} \leq \mathbf{amls}(\alpha, c)^n \cdot n^{\mathcal{O}(1)} \end{aligned}$$

which results in the running time stated in the theorem. \blacktriangleleft

5 The Brute-Force Approximation Algorithm

In this section we describe an α -approximate variant of exhaustive search that runs in time $\mathcal{O}^*((\mathbf{brute}(\alpha))^n)$, where $\mathbf{brute}(\alpha) = 1 + \exp(-\alpha \cdot \mathcal{H}(\frac{1}{\alpha}))$. We complement this result by showing that $\mathcal{O}^*((\mathbf{brute}(\alpha))^n)$ is the best possible running time of an α -approximation algorithm for a subset minimization problem in the *oracle model* defined below.

A (randomized) *oracle α -approximation minimum subset* algorithm takes as input a universe U and receives a membership oracle to a *monotone* family $\mathcal{F} \subseteq 2^U$. The algorithm returns a set $S \in \mathcal{F}$ such that $|S| \leq \alpha \cdot \min \{|T| \mid T \in \mathcal{F}\}$ (with constant probability).

► **Theorem 5.1.** *For any $\alpha \geq 1$, there is a deterministic oracle α -approximation minimum subset algorithm which runs in time $\mathcal{O}^*((\mathbf{brute}(\alpha))^n)$. Moreover, there is no randomized oracle α -approximation minimum subset algorithm which uses $\mathcal{O}^*(c^n)$ oracle queries for any $c < \mathbf{brute}(\alpha)$.*

The proof of Theorem 5.1 utilizes the technical bound proved in Lemma 5.2. The proof of Lemma 5.2 uses arguments that similar to the ones used in the proof of Lemma 3.4.

► **Lemma 5.2 (*)**. *For any $n \in \mathbb{N}$ and $\alpha \geq 1$ it holds that*

$$n^{-\mathcal{O}(1)} \cdot (\mathbf{brute}(\alpha))^n \leq \max_{k \in [0, \frac{n}{\alpha}] \cap \mathbb{N}} \frac{\binom{n}{k}}{\binom{\lfloor \alpha k \rfloor}{k}} \leq n^{\mathcal{O}(1)} \cdot (\mathbf{brute}(\alpha))^n$$

To obtain the claimed algorithm of Theorem 5.1 the basic idea is to sample $\mathcal{O}^*((\mathbf{brute}(\alpha))^n)$ random sets (of some size k) and show that the desired approximate solution is found with constant probability. This algorithm can be derandomized by using

Theorem 4.2 for the special case when $p = r$. However, this introduces another sublinear term in the exponent of the running time. Instead, in this special case, we can rely on existing results on covering families [19].

Let $k < t < n$ be natural numbers and recall $[n] := \{1, 2, \dots, n\}$. An (n, t, k) -covering is a family $\mathcal{C} \subseteq \{X \mid X \subseteq [n], |X| = t\}$ such that, for every $S \subseteq [n]$ of size $|S| = k$, there is some $X \in \mathcal{C}$ such that $S \subseteq X$. To construct the algorithm for Theorem 5.1, we exploit known constructions of (n, t, k) -coverings of almost optimal size.

► **Theorem 5.3** (Kuzjurin [19]). *There is an algorithm that, given $k < t < n$, computes an (n, t, k) -covering \mathcal{C} of size $(1 + o(1)) \cdot \binom{n}{k} / \binom{t}{k}$ in time $|\mathcal{C}| \cdot n^{\mathcal{O}(1)}$.*

Proof of Theorem 5.1. We first show the algorithmic part. By renaming elements, we may assume $U = [n]$. For every $k \leq n/\alpha$ we compute a $(n, \lfloor \alpha k \rfloor, k)$ -covering \mathcal{C}_k using Theorem 5.3 and check for every set $X \in \mathcal{C}_k$ whether it is contained in \mathcal{F} using the oracle. We return the smallest set that is contained in \mathcal{F} . If no such set is found, we return the entire universe.

It is easy to see that this algorithm is an α -approximation algorithm. Indeed, let $\text{OPT} \subseteq U$ be a solution set of minimum cardinality and let $k := |\text{OPT}|$. If $k \geq n/\alpha$, then the algorithm is clearly correct since even returning the entire universe gives the desired approximation ratio. So suppose that $k \leq n/\alpha$. By definition of an $(n, \lfloor \alpha k \rfloor, k)$ -covering there is some $X \in \mathcal{C}_k$ such that $\text{OPT} \subseteq X$ and $|X| = \lfloor \alpha k \rfloor \leq \alpha k$. Since \mathcal{F} is monotone we get that $X \in \mathcal{F}$ and the algorithm returns a solution set of size at most $|X| \leq \alpha k$.

By Theorem 5.3, the algorithm runs in time $\max_{k \in [0, \frac{n}{\alpha}] \cap \mathbb{N}} \binom{n}{k} / \binom{\lfloor \alpha k \rfloor}{k} \cdot n^{\mathcal{O}(1)}$. From Lemma 5.2 we get that the running time is bounded by $(\text{brute}(\alpha))^n \cdot n^{\mathcal{O}(1)}$.

Next, we prove the lower bound. For $n \in \mathbb{N}$ define $\kappa(n) = \arg\max_{k \in [0, \frac{n}{\alpha}] \cap \mathbb{N}} \binom{n}{k} / \binom{\lfloor \alpha k \rfloor}{k}$. For every $n \in \mathbb{N}$ we define $\mathcal{F}_{\text{adv}}(n) = \{S \subseteq [n] \mid |S| > \alpha \kappa(n)\}$. Also, for every $n \in \mathbb{N}$ and $X \subseteq [n]$ we define $\mathcal{F}(n, X) = \{S \subseteq [n] \mid X \subseteq S\} \cup \mathcal{F}_{\text{adv}}(n)$. It can be easily observed that $\mathcal{F}(n, X)$ and $\mathcal{F}_{\text{adv}}(n)$ are monotone set families. Our bound is based on the difficulty that algorithms have to distinguish between $\mathcal{F}_{\text{adv}}(n)$ and $\mathcal{F}(n, X)$.

Let \mathcal{A} be a randomized oracle α -approximation minimum subset algorithm. Without loss of generality we assume \mathcal{A} only returns a set S if it queried the oracle with that set (and got back a positive answer). Let $q(n)$ be the maximal number of oracle queries the algorithm uses given a universe of size n . As the algorithm is randomized, we use R to denote the random sequence of bits used by the algorithm.

Fix $n \in \mathbb{N}$. For any $j \in [q(n)]$ the j -th query to the oracle is a function of the previous answers the algorithm received from the oracle and the sequence of random bits the algorithm uses. Thus, there is a function $S_j(R)$ which returns the j -th query the algorithm sends to the oracle, given that the algorithm gets an oracle to $\mathcal{F}_{\text{adv}}(n)$. If the algorithm does not issue the j -th query given R we arbitrarily define $S_j(R) = \emptyset$.

Let $X \subseteq [n]$ be a random set of size $\kappa(n)$ which is sampled uniformly (and independently of R). Consider the execution of \mathcal{A} with the universe $[n]$ and an oracle for $\mathcal{F}(n, X)$. Define

$$\mathcal{C}(R) = \bigcup_{j=1}^{q(n)} \begin{cases} \{T \subseteq [n] \mid |T| = \kappa(n), T \subseteq S_j(R)\} & \text{if } |S_j(R)| \leq \alpha \cdot \kappa(n) \\ \emptyset & \text{otherwise} \end{cases}. \quad (8)$$

If $X \notin \mathcal{C}(R)$ then the answers the algorithm receives to its queries are identical to the answers it would have received if it was given an oracle $\mathcal{F}_{\text{adv}}(n)$. It therefore asks the same queries, and must return a set $S \in \mathcal{F}_{\text{adv}}(n)$. As \mathcal{A} is a randomized α -approximation algorithm, there is $\gamma \in (0, 1]$ such that \mathcal{A} returns a set $S \in \mathcal{F}(n, X)$ which satisfies $|S| \leq \alpha \cdot |X| = \alpha \cdot \kappa(n)$ with probability at least γ . As all the sets in $\mathcal{F}_{\text{adv}}(n)$ have cardinality greater than $\alpha \cdot \kappa(n)$ it follows that $\Pr(X \notin \mathcal{C}(R)) \leq 1 - \gamma$, or equivalently, $\Pr(X \in \mathcal{C}(R)) \geq \gamma$.

50:16 Approximate Monotone Local Search

By the definition of $\mathcal{C}(R)$ (8), each query $S_j(R)$ adds at most $\binom{\lfloor \alpha \cdot \kappa(n) \rfloor}{\kappa(n)}$ sets to $\mathcal{C}(R)$. Thus $|\mathcal{C}(R)| \leq q(n) \cdot \binom{\lfloor \alpha \cdot \kappa(n) \rfloor}{\kappa(n)}$. Since X is independent of R we have,

$$\gamma \leq \Pr(X \in \mathcal{C}(R)) \leq \frac{q(n) \cdot \binom{\lfloor \alpha \cdot \kappa(n) \rfloor}{\kappa(n)}}{\binom{n}{\kappa(n)}},$$

and hence

$$q(n) \geq \gamma \cdot \frac{\binom{n}{\kappa(n)}}{\binom{\lfloor \alpha \cdot \kappa(n) \rfloor}{\kappa(n)}} = \gamma \cdot \max_{k \in [0, \frac{n}{\alpha}] \cap \mathbb{N}} \frac{\binom{n}{k}}{\binom{\lfloor \alpha k \rfloor}{k}} \geq n^{-\mathcal{O}(1)} \cdot (\mathbf{brute}(\alpha))^n,$$

where the equality follows from the definition of $\kappa(n)$ and the last inequality follows from Lemma 5.2. In particular, \mathcal{A} does not use $\mathcal{O}^*(c^n)$ oracle queries for any $c < \mathbf{brute}(\alpha)$. ◀

6 Concluding Remarks

We introduced and analyzed approximate monotone local search **Approximate-MLS** which can be used to obtain faster exponential approximation algorithms from parameterized (extension) approximation algorithms for monotone subset minimization problems. In particular, we obtain faster exponential approximation algorithms for **VERTEX COVER**, **3-HITTING SET**, **DFVS**, **SUBSET DFVS**, **DOCT** and **UNDIRECTED MULTICUT** (for some approximation ratios).

Following the submission of this work we became aware of a similar application of monotone local search in the PhD thesis of Lee [20]. However, Lee mainly provides experimental evaluations of the running time of approximate monotone local search for specific problems such as **VERTEX COVER** and **FEEDBACK VERTEX SET**, and does not provide a rigorous analysis of the general running time which is the main focus of this work.

The significance of **Exact-MLS** stems from the abundance of existing parameterized (extension) algorithms which can be used to obtain the state-of-art exponential algorithms for multiple problems. **Approximate-MLS** has a similar potential in the context of exponential approximation algorithms. Thus, our result further emphasizes the importance of the already-growing field of parameterized approximation, by exhibiting its strong connections with exponential-time approximations.

Some interesting follow-up questions of our work are the following.

► **Problem 6.1.** *Can an α -approximate algorithm running in time $\mathcal{O}^*((\mathbf{brute}(\alpha) - \varepsilon)^n)$ be derived from a parameterized extension β -approximation algorithm for any $\beta > \alpha$?*

For example, for **DIRECTED FEEDBACK VERTEX SET** only a parameterized 2-approximation algorithm running in time $\mathcal{O}^*(c^k)$ [22] is currently available. The question is whether this algorithm can also be used to obtain an exponential 1.1-approximation algorithm that runs in time $\mathcal{O}^*((\mathbf{brute}(1.1) - \varepsilon)^n)$, for some $\varepsilon > 0$?

We also described and showed that the exhaustive search analog in the α -approximate setting achieves the best possible running time of $\mathcal{O}^*((\mathbf{brute}(\alpha))^n)$ when one only has access to a membership oracle for the problem. Observe that for $\alpha = 1$, **SETH** asserts that $(\mathbf{brute}(1))^n = 2^n$ is tight.

► **Problem 6.2.** *Does there exist a monotone subset minimization problem for which there is no α -approximation algorithm that runs in time $\mathcal{O}^*((\mathbf{brute}(\alpha) - \varepsilon)^n)$, assuming **SETH**?*

Recall that the `Approximate-MLS` algorithm only uses random sampling and the given parameterized α -approximation extension algorithm. Another interesting lower bound question is the following.

► **Problem 6.3.** *Can one show that the running time of `Approximate-MLS` is tight (up to polynomial factors) when one is only given access to a membership oracle and a parameterized α -approximation extension algorithm as a black-box?*

References

- 1 Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *J. ACM*, 62(5):42:1–42:25, 2015. doi:10.1145/2775105.
- 2 Nikhil Bansal, Parinya Chalermsook, Bundit Laekhanukit, Danupon Nanongkai, and Jesper Nederlof. New tools and connections for exponential-time approximation. *Algorithmica*, 81(10):3993–4009, 2019. doi:10.1007/s00453-018-0512-8.
- 3 Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Approximation of max independent set, min vertex cover and related problems by moderately exponential algorithms. *Discret. Appl. Math.*, 159(17):1954–1970, 2011. doi:10.1016/j.dam.2011.07.009.
- 4 Ljiljana Brankovic and Henning Fernau. Parameterized approximation algorithms for hitting set. In Roberto Solis-Oba and Giuseppe Persiano, editors, *Approximation and Online Algorithms – 9th International Workshop, WAOA 2011, Saarbrücken, Germany, September 8-9, 2011, Revised Selected Papers*, volume 7164 of *Lecture Notes in Computer Science*, pages 63–76. Springer, 2011. doi:10.1007/978-3-642-29116-6_6.
- 5 Ljiljana Brankovic and Henning Fernau. A novel parameterised approximation algorithm for minimum vertex cover. *Theor. Comput. Sci.*, 511:85–108, 2013. doi:10.1016/j.tcs.2012.12.003.
- 6 Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, 2nd edition, 2006. doi:10.1002/047174882X.
- 7 Marek Cygan, Lukasz Kowalik, and Mateusz Wykucz. Exponential-time approximation of weighted set cover. *Inf. Process. Lett.*, 109(16):957–961, 2009. doi:10.1016/j.ipl.2009.05.003.
- 8 Pavel Dvorák, Andreas Emil Feldmann, Dusan Knop, Tomáš Masarík, Tomáš Toufar, and Pavel Veselý. Parameterized approximation schemes for Steiner trees with small number of Steiner vertices. *SIAM J. Discret. Math.*, 35(1):546–574, 2021. doi:10.1137/18M1209489.
- 9 Barış Can Esmer, Ariel Kulik, Dániel Marx, Daniel Neuen, and Roohani Sharma. Faster exponential-time approximation algorithms using approximate monotone local search. *CoRR*, abs/2206.13481, 2022. doi:10.48550/arXiv.2206.13481.
- 10 Uriel Feige and Mohammad Mahdian. Finding small balanced separators. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 375–384. ACM, 2006. doi:10.1145/1132516.1132573.
- 11 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/a13060146.
- 12 Michael R. Fellows, Ariel Kulik, Frances A. Rosamond, and Hadas Shachnai. Parameterized approximation via fidelity preserving transformations. *J. Comput. Syst. Sci.*, 93:30–40, 2018. doi:10.1016/j.jcss.2017.11.001.
- 13 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *J. ACM*, 66(2):8:1–8:23, 2019. doi:10.1145/3284176.
- 14 Anupam Gupta, Euiwoong Lee, and Jason Li. Faster exact and approximate algorithms for k-cut. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 113–123. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00020.

- 15 Anupam Gupta, Euiwoong Lee, and Jason Li. An FPT algorithm beating 2-approximation for k-cut. 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 2821–2837. SIAM, 2018. doi:10.1137/1.9781611975031.179.
- 16 Ken-ichi Kawarabayashi and Bingkai Lin. A nearly $5/3$ -approximation FPT algorithm for min-k-cut. 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 990–999. SIAM, 2020. doi:10.1137/1.9781611975994.59.
- 17 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.
- 18 Ariel Kulik and Hadas Shachnai. Analysis of two-variable recurrence relations with application to parameterized approximations. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 762–773. IEEE, 2020. doi:10.1109/FOCS46700.2020.00076.
- 19 Nikolai N. Kuzjurin. Explicit constructions of Rödl’s asymptotically good packings and coverings. *Comb. Probab. Comput.*, 9(3):265–276, 2000. doi:10.1017/S0963548300004235.
- 20 Edward Lee. *Exponential time algorithms via separators and random subsets*. PhD thesis, University of New South Wales, 2021. doi:10.26190/unsworks/22740.
- 21 Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Math. Program.*, 177(1-2):1–19, 2019. doi:10.1007/s10107-018-1255-7.
- 22 Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. FPT-approximation for FPT problems. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 199–218. SIAM, 2021. doi:10.1137/1.9781611976465.14.
- 23 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min k-cut. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 798–809. IEEE, 2020. doi:10.1109/FOCS46700.2020.00079.
- 24 Pasin Manurangsi. A note on max k-vertex cover: Faster fpt-as, smaller approximate kernel and improved approximation. 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 15:1–15:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASICs.SOSA.2019.15.
- 25 Pasin Manurangsi and Luca Trevisan. Mildly exponential time approximation algorithms for vertex cover, balanced separator and uniform sparsest cut. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 – Princeton, NJ, USA*, volume 116 of *LIPICs*, pages 20:1–20:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.APPROX-RANDOM.2018.20.
- 26 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.
- 27 Dániel Marx and Igor Razgon. Constant ratio fixed-parameter approximation of the edge multicut problem. *Inf. Process. Lett.*, 109(20):1161–1166, 2009. doi:10.1016/j.ipl.2009.07.016.
- 28 Igor Razgon. Computing minimum directed feedback vertex set in $O^*(1.9977^n)$. In Giuseppe F. Italiano, Eugenio Moggi, and Luigi Laura, editors, *Theoretical Computer Science, 10th Italian Conference, ICTCS 2007, Rome, Italy, October 3-5, 2007, Proceedings*, pages 70–81. World Scientific, 2007. doi:10.1142/9789812770998_0010.
- 29 Piotr Skowron and Piotr Faliszewski. Chamberlin-Courant rule with approval ballots: approximating the MaxCover problem with bounded frequencies in FPT time. *J. Artificial Intelligence Res.*, 60:687–716, 2017. doi:10.1613/jair.5628.

- 30 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001. doi:10.1007/978-3-662-04565-7.

Intersection Searching Amid Tetrahedra in 4-Space and Efficient Continuous Collision Detection

Esther Ezra  

School of Computer Science, Bar Ilan University, Ramat Gan, Israel

Micha Sharir  

School of Computer Science, Tel Aviv University, Tel Aviv, Israel

Abstract

We develop data structures for intersection detection queries in four dimensions that involve segments, triangles and tetrahedra. Specifically, we study two main problems: (i) Preprocess a set of n tetrahedra in \mathbb{R}^4 into a data structure for answering segment-intersection queries amid the given tetrahedra (referred to as *segment-tetrahedron intersection queries*), and (ii) Preprocess a set of n triangles in \mathbb{R}^4 into a data structure that supports triangle-intersection queries amid the input triangles (referred to as *triangle-triangle intersection queries*). As far as we can tell, these problems have not been previously studied.

For problem (i), we first present a “standard” solution which, for any prespecified value $n \leq s \leq n^6$ of a so-called storage parameter s , yields a data structure with $O^*(s)$ storage and expected preprocessing, which answers an intersection query in $O^*(n/s^{1/6})$ time (here and in what follows, the $O^*(\cdot)$ notation hides subpolynomial factors). For problem (ii), using similar arguments, we present a solution that has the same asymptotic performance bounds.

We then improve the solution for problem (i), and present a more intricate data structure that uses $O^*(n^2)$ storage and expected preprocessing, and answers a segment-tetrahedron intersection query in $O^*(n^{1/2})$ time. Using the parametric search technique of Agarwal and Matoušek [3], we can obtain data structures with similar performance bounds for the *ray-shooting* problem amid tetrahedra in \mathbb{R}^4 . Unfortunately, so far we do not know how to obtain a similar improvement for problem (ii).

Our algorithms are based on a primal-dual technique for range searching with semi-algebraic sets, based on recent advances in this area [2, 11]. As this is a result of independent interest, we spell out the details of this technique.

As an application, we present a solution to the problem of “continuous collision detection” amid moving tetrahedra in 3-space. That is, the workspace consists of n tetrahedra, each moving at its own fixed velocity, and the goal is to detect a collision between some pair of moving tetrahedra. Using our solutions to problems (i) and (ii), we obtain an algorithm that detects a collision in $O^*(n^{12/7})$ expected time. We also present further applications, including an output-sensitive algorithm for constructing the arrangement of n tetrahedra in \mathbb{R}^4 and an output-sensitive algorithm for constructing the intersection or union of two or several nonconvex polyhedra in \mathbb{R}^4 .

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Computational geometry, Ray shooting, Tetrahedra in \mathbb{R}^4 , Intersection queries in \mathbb{R}^4 , Polynomial partitioning, Range searching, Semi-algebraic sets, Tradeoff

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.51

Related Version *Full Version*: <http://arxiv.org/abs/2208.06703>

Funding *Esther Ezra*: Work partially supported by NSF CAREER under Grant CCF:AF-1553354 and by Grant 824/17 from the Israel Science Foundation.

Micha Sharir: Work partially supported by Grant 260/18 from the Israel Science Foundation.



© Esther Ezra and Micha Sharir;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 51; pp. 51:1–51:17

Leibniz International Proceedings in Informatics



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

1 Introduction

In this paper we consider various intersection problems involving segments, triangles and tetrahedra in \mathbb{R}^4 . In four dimensions, the interesting setups involve (i) intersections between (one-dimensional) segments and (three-dimensional) tetrahedra, and (ii) between (two-dimensional) triangles and (two-dimensional) triangles. We study both problems, and derive efficient solutions to each of them.

As an interesting application, we consider the *continuous collision detection* problem, where the input consists of n tetrahedra in \mathbb{R}^3 , each of which is moving at some constant velocity of its own, and the goal is to detect whether any pair of them collide. Adding the time as a fourth coordinate, this becomes a batched version of intersection detection in \mathbb{R}^4 , involving both setups (i) and (ii). Other applications include output-sensitive construction of the arrangement of n tetrahedra in \mathbb{R}^4 , and an output-sensitive algorithm for computing the intersection or the union of two or several not necessarily convex polyhedra in \mathbb{R}^4 . In the three-dimensional versions of these problems, which were recently studied in [7], the only setup that needed to be considered was segment intersection amid triangles. In four dimensions, though, we also face the triangle-triangle intersection problem, since we also need to find intersections between pairs of 2-faces of the input objects.

Setup (i): Segment-tetrahedron intersection queries. Consider first the case of segments vs. tetrahedra. In the setup considered here, the input objects are n (not necessarily pairwise openly disjoint) tetrahedra in \mathbb{R}^4 and the query objects are segments, and the goal is to detect, count, or report intersections between the query segment and the input tetrahedra.

As far as we can tell, this problem has not been explicitly studied so far. We first present, in Section 2, a “traditional” (albeit novel) solution, in which the problem is reduced to a range searching problem in a suitable parametric space, which, in the case of (lines supporting) segments in \mathbb{R}^4 , is six-dimensional. We carefully adapt and combine recent techniques, developed by Agarwal et al. [2] and Matoušek and Patáková [11], which provide algorithmic constructions of intricate space decompositions based on partitioning polynomials. Using this machinery, we solve the problem so that, with a so-called storage parameter s , a segment intersection query can be answered in¹ $O^*(n/s^{1/6})$ time, for any $n \leq s \leq n^6$, and the storage and preprocessing cost are both $O^*(s)$.

A special case of this setup is an extension to four dimensions of the classical *ray shooting* problem, which has mostly been studied in two and three dimensions. In a general setting, we are given a collection S of n simply-shaped objects, and the goal is to preprocess S into a data structure that supports efficient ray shooting queries, where each query specifies a ray ρ and asks for the first object of S hit by ρ , if such an object exists. In this work we only consider the (already challenging) case of input tetrahedra. Using the parametric search technique of Agarwal and Matoušek [3], ray shooting queries can be reduced to segment-intersection detection queries, up to a polylogarithmic factor in the query cost. By the above discussion, we obtain the following result:

► **Theorem 1.** *Given a collection \mathcal{T} of n tetrahedra in \mathbb{R}^4 , and any storage parameter $n \leq s \leq n^6$, we can preprocess \mathcal{T} into a data structure of size $O^*(s)$, in randomized $O^*(s)$ expected time, so that we can answer any segment-intersection or ray-shooting query in \mathcal{T} in $O^*(n/s^{1/6})$ time. The query time bound applies to segment-intersection detection and counting queries (and to ray shooting queries). The cost is $O^*(n/s^{1/6}) + O(k)$ for reporting queries.*

¹ As in the abstract, the $O^*(\cdot)$ notation hides subpolynomial factors, typically of the form n^ε , for any $\varepsilon > 0$, and their coefficients which depend on ε .

We later improve upon this standard algorithm in Section 4, where we show:

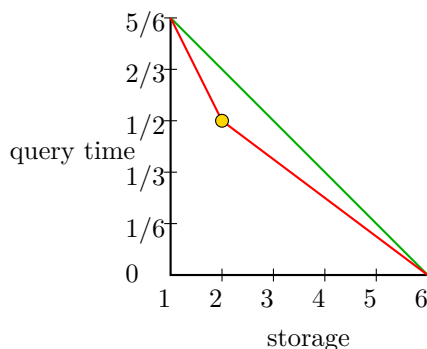
► **Theorem 2.** *A collection S of n arbitrary tetrahedra in \mathbb{R}^4 can be preprocessed into a data structure of size $O^*(n^2)$, in expected time $O^*(n^2)$, which supports segment-intersection detection and counting queries and ray-shooting queries in time $O^*(n^{1/2})$ per query.*

This indeed improves the bounds stated in Theorem 1, which, with $s = O^*(n^2)$ storage, has query time $O^*(n^{2/3})$. Furthermore, with the storage bound specified in Theorem 2, the query bound is similar to that obtained for ray-shooting amid hyperplanes (rather than tetrahedra) in \mathbb{R}^4 [3].

We then go on, in the full version of the paper² to extend the result to obtain a tradeoff between storage (and expected preprocessing time) and query time. We show that, with storage parameter s , which can vary between n and n^6 , we can answer a segment intersection or a ray shooting query in time

$$Q(n, s) = \begin{cases} O^*\left(\frac{n^{7/6}}{s^{1/3}}\right) & \text{for } s = O(n^2) \\ O^*\left(\frac{n^{3/4}}{s^{1/8}}\right) & \text{for } s = \Omega(n^2). \end{cases} \quad (1)$$

See Figure 1 for an illustration. This yields algorithms that answer m segment intersection or ray-shooting queries on n tetrahedra in $\max\{O^*(m^{3/4}n^{7/8} + n), O^*(m^{8/9}n^{2/3} + m)\}$ time and storage. The first (resp., second) bound dominates when $m \leq n^{3/2}$ (resp., $m \geq n^{3/2}$).



■ **Figure 1** The tradeoff between storage and query time. The breakpoint in the graph represents the case studied in Section 4. Both axes are drawn on a logarithmic scale.

Setup (ii): Triangle-triangle intersection detection. We next consider the other setup of intersection queries, where both input and query objects are triangles in \mathbb{R}^4 . We show that this setup can also be reduced, similar to setup (i), to a multi-level range searching problem in \mathbb{R}^6 involving semi-algebraic ranges. This allows us to obtain the same performance bounds here too. Namely we have:

► **Theorem 3.** *Given a collection Δ of n triangles in \mathbb{R}^4 , and any storage parameter $n \leq s \leq n^6$, we can preprocess Δ into a data structure of size $O^*(s)$, in randomized $O^*(s)$ expected time, so that we can answer any triangle-intersection query in Δ in $O^*(n/s^{1/6})$ time.*

² Soon to be available on arXiv.

Since both input and query objects are triangles, it is also interesting to consider the bichromatic batched version of the problem. Namely we have:

► **Theorem 4.** *Given two collections R and B of triangles in \mathbb{R}^4 , of respective sizes m and n , We can detect an intersection between some triangle of R and some triangle of B , or count all such intersections, in time $O^*(m^{6/7}n^{6/7} + m + n)$. We can also report all these intersections in time $O^*(m^{6/7}n^{6/7} + m + n + k)$, where k is the output size.*

As a consequence, integrating this bound with the one obtained in Theorem 1 (in which, similar to the preceding argument, we need to set $s = n^{12/7}$ to match the performance with that stated above, as is easily verified), we obtain an overall $O^*(n^{12/7})$ expected time solution for the continuous collision detection problem, that is:³

► **Theorem 5.** *Given n tetrahedra in \mathbb{R}^3 , each of which is moving at some constant velocity of its own, one can detect a collision between any pair of moving tetrahedra in $O^*(n^{12/7})$ expected time.*

Collision detection has been widely studied – see Lin, Manocha and Kim [10] for a recent comprehensive survey, and the references therein. We are not aware of any work that addresses the exact algorithmic approach for the specific setup considered here, although there are some works, such as Canny [6] or Schömer and Thiel [12], that address similar contexts.

We then consider the applications of our techniques to the problems of output-sensitive construction of an arrangement of tetrahedra in \mathbb{R}^4 , and of constructing the intersection or union of two or several (nonconvex) polyhedra in \mathbb{R}^4 . Using the bounds for setups (i) and (ii), we obtain, in Section 5:

► **Theorem 6.** *(i) Let \mathcal{T} be a collection of n tetrahedra in general position in \mathbb{R}^4 . We can construct the arrangement $\mathcal{A}(\mathcal{T})$ of \mathcal{T} in $O^*(n^{12/7} + n^{1/2}k_2 + k_4)$ expected time, where k_2 is the number of intersecting pairs of tetrahedra in \mathcal{T} , and k_4 is the number of vertices of $\mathcal{A}(\mathcal{T})$. (ii) Given two arbitrary polyhedra R and B in \mathbb{R}^4 , each of complexity $O(n)$, that lie in general position with respect to one another, the intersection $R \cap B$ can be computed in expected time $O^*(n^{12/7} + n^{1/2}k_2 + k_4)$, where k_2 is the number of 2-faces of $\mathcal{A}(R \cup B)$, and k_4 is the number of vertices of $\mathcal{A}(R \cup B)$.*

As another application of our technique we present, in the full version, an efficient algorithm for detecting or reporting intersections between n 2-flats and n lines in \mathbb{R}^4 . We show that, given n lines and n 2-flats in \mathbb{R}^4 , one can detect whether some line intersects some 2-flat in $O^*(n^{13/8})$ expected time. One can also report all k intersections in $O^*(n^{13/8} + k)$ expected time. This result is a degenerate special case of the triangle-triangle intersection setup, and admits a faster solution. (Note that in general position 2-flats and lines are not expected to meet in \mathbb{R}^4 , which makes this special case interesting.)

Setup (iii): Tetrahedron-segment intersection queries. We can also handle a symmetric setup, in which the input consists of n segments in \mathbb{R}^4 and the query is with a tetrahedron T , where the goal is to detect, count or report intersections between T and the input segments. Using a similar machinery, we obtain the same asymptotic performance bounds, as in the standard solution, for this setup too.

³ Here we use an obvious divide-and-conquer approach in order to reduce the general (non-bichromatic) problem to the bichromatic version.

► **Theorem 7.** *Given a collection S of n segments in \mathbb{R}^4 , and any storage parameter $n \leq s \leq n^6$, we can preprocess S into a data structure of size $O^*(s)$, in randomized $O^*(s)$ expected time, so that we can answer any tetrahedron-intersection query in S in $O^*(n/s^{1/6})$ time. The query time bound applies to tetrahedron-intersection detection and counting queries. The cost is $O^*(n/s^{1/6}) + O(k)$ for reporting queries.*

2 Segment-Intersection amid Tetrahedra: An Initial Algorithm

In this section we present an initial solution to the problem of segment-intersection detection amid tetrahedra in four dimensions, which is based on a careful combination of the recent range searching machinery of [2, 11]. We do so because (a) as far as we can tell, such a solution has not yet been spelled out in the literature, (b) the adaptation of the available techniques to this problem is not simple, requires nontrivial and careful enhancements, and is of independent interest, and (c) this gives a yardstick for appreciating the improvement obtained by our improved algorithm, presented in Section 4.

The parametric search technique of Agarwal and Matoušek [3] reduces ray shooting queries to segment-intersection detection queries, so it suffices to consider the latter problem. The reporting and counting variants are simple extensions of the same technique, as will be discussed as we go.

To obtain a tradeoff between the storage of the structure and the query time, our algorithm uses a primal-dual approach. However, both the primal and dual setups suffer from the fact that, in four dimensions, segments and tetrahedra require too many parameters to specify. Specifically, a segment requires eight parameters (e.g., by specifying its two endpoints), while a tetrahedron requires 16 parameters (e.g., by specifying the coordinates of its four vertices).

To address this issue, we use a multi-level data structure, where each level caters to one aspect of the condition that a segment crosses a tetrahedron. This is done so that, at each of these levels, the number of parameters that a segment or a tetrahedron requires is at most six. Specifically, the condition that a segment e , that lies on a line ℓ , intersects a tetrahedron Δ , supported by a hyperplane h_Δ , is the conjunction of the following conditions:

- (i) The two endpoints of e lie on different sides of h_Δ .
- (ii) With a suitable choice of a direction of ℓ and an orientation of Δ , ℓ has a positive orientation with respect to each of the 2-planes that support the four 2-faces of Δ .

Conditions (i) and (ii) are the conjunction of a total of six sub-conditions: the first two conditions tests the position of some endpoint of e with respect to the hyperplanes h_Δ , and the other four conditions tests the orientation of ℓ with respect to the 2-planes supporting specific 2-faces of the tetrahedra. Thus, the dual structure has six levels, two for testing the sub-conditions of condition (i) and four for testing the sub-conditions of condition (ii).

More precisely, each but the last level collects all the tetrahedra Δ that satisfy the corresponding sub-condition for the query segment (that a specific endpoint of e lies in a specific side of h_Δ for the first two levels, and that the oriented 2-plane supporting a specific 2-face of Δ is positively oriented with respect to the directed line ℓ for the last four levels), as the disjoint union of precomputed canonical sets of tetrahedra. The last level just tests whether the last sub-condition is satisfied for any tetrahedron in the current canonical set.

We use the fact that lines in \mathbb{R}^4 require six real parameters to specify. The space of lines in \mathbb{R}^4 is actually projective, but for simplicity of presentation we regard it as a real space, and ignore the special cases in which the real representation fails. Handling these cases follows the same approach, and is in fact simpler. Alternatively, a generic (say random) rotation of the coordinate frame allows us to ignore them altogether.

One simple way to represent a line ℓ in \mathbb{R}^4 is by the points $u_\ell^0 = (x_0, y_0, z_0, 0)$ and $u_\ell^1 = (x_1, y_1, z_1, 1)$ at which ℓ crosses the hyperplanes $w = 0$ and $w = 1$, respectively (ignoring lines that are orthogonal to the w -axis), so the line ℓ can be represented as the point $p_\ell = (x_0, y_0, z_0, x_1, y_1, z_1)$ in \mathbb{R}^6 , as desired.

Similarly, 2-planes in \mathbb{R}^4 also require six parameters to specify. This is simply because the duality in \mathbb{R}^4 maps lines to 2-planes and vice versa, but a concrete way to represent 2-planes by six parameters is to specify three points on a 2-plane π that are intersections of π with three fixed 2-planes, such as, say, $x = y = 0$, $x = 0$ and $y = 1$, and $x = y = 1$ (again ignoring special directions of π). Each of the intersection points has two degrees of freedom (as two of its coordinates are fixed), for a total of six. Denote these points as $v_\pi^{(00)}$, $v_\pi^{(01)}$, and $v_\pi^{(11)}$, and put $q_\pi = (v_\pi^{(00)}, v_\pi^{(01)}, v_\pi^{(11)})$, listing only the w - and z -coordinates of each point, so q_π is a point in \mathbb{R}^6 .

These observations are meaningful only for the last four levels of the structure. The first two levels are simpler, as they deal with points (the endpoints of e) and hyperplanes (those supporting the tetrahedra of \mathcal{T}) in \mathbb{R}^4 . Thus each of the first two levels is a halfspace range searching structure for points and halfspaces in \mathbb{R}^4 . (Actually, this is the case when we pass to the dual 4-space; in the primal we have a point-enclosure problem, where the query is a point and the input consists of halfspaces bounded by the relevant hyperplanes.) Using standard techniques (see, e.g., [1]), this can be done, for N halfspaces in the current canonical subset and using $O^*(N)$ storage, so that a query costs $O^*(N^{3/4})$ time.⁴ This cost will be subsumed by the query time bounds for the last four levels. The cost of a query includes the cost of reporting its output, as a list of canonical sets.

We next consider the (more involved) situation in the last four levels of the structure. Here the query segment is replaced by its supporting line ℓ , and each tetrahedron Δ is replaced by the 2-plane supporting a specific 2-face of Δ . In the primal setup, the line ℓ is represented as a point p_ℓ in (projective) 6-space, in the manner just described, and a tetrahedron Δ , represented by a suitable 2-plane π , is represented as a semi-algebraic region K_π , consisting of all points that represent (directed) lines that are positively oriented with respect to π . The problem that we face is a point-enclosure query, in which we want to determine whether p_ℓ lies in any of the regions K_π (alternatively, count or report all these regions). In the dual setup, the 2-planes π are represented as points in \mathbb{R}^6 , and the (directed) query line ℓ is represented as a semi-algebraic region Q_ℓ that consists of all (oriented) 2-planes that are positively oriented with respect to ℓ . The problem here is a semi-algebraic range searching query, where we want to determine whether Q_ℓ contains any input point (alternatively, count or report all these points).

The orientation test of ℓ with respect to π amounts to computing the sign of the 5×5 determinant

$$\begin{vmatrix} u_\ell^0 & 1 \\ u_\ell^1 & 1 \\ v_\pi^{(00)} & 1 \\ v_\pi^{(01)} & 1 \\ v_\pi^{(11)} & 1 \end{vmatrix}, \quad (2)$$

with a suitable orientation of the pair of points u_ℓ^0, u_ℓ^1 on ℓ (dictating the direction of ℓ), and of the triple of points $v_\pi^{(00)}, v_\pi^{(01)}, v_\pi^{(11)}$ on π (dictating the orientation of π).

⁴ A tradeoff between storage and query time is also available, but we do not need it here.

To compute these signs, at each of the four latter levels of the structure, we use a primal-dual approach, where the top part of the structure is in the primal, and at each of its leaf nodes we pass to the dual.

The dual setup. The dual setup is simpler, so we begin with its description. In the dual setup, each tetrahedron Δ of the current canonical subset of \mathcal{T} is mapped to the point $q_\pi = (v_\pi^{(00)}, v_\pi^{(01)}, v_\pi^{(11)})$ in \mathbb{R}^6 , where π is the 2-plane supporting the 2-face of Δ that corresponds to the present level. The query line ℓ is mapped to a semi-algebraic region Q_ℓ of constant complexity in \mathbb{R}^6 , consisting of all points q_π that represent (oriented) 2-planes that have positive orientation with respect to ℓ , that is, the corresponding determinant in (2) is positive. (The resulting polynomial is cubic in q_π .)

As already mentioned, the task at hand, at each but the last level, is to collect the points q_π that lie in Q_ℓ , as the disjoint union of a small number of precomputed canonical sets of tetrahedra, and the task at the last level is to determine whether Q_ℓ contains any point q_π , for π corresponding to the last 2-faces of the tetrahedra in the present canonical subset of \mathcal{T} . In other words, we have, at each of these levels, a problem involving range searching with semi-algebraic ranges in \mathbb{R}^6 . Using the algorithm of Matoušek and Patáková [11], which is a simplified version of the algorithm of Agarwal et al. [4], this can be done, for N tetrahedra with $O^*(N)$ storage, so that a query takes $O^*(N^{5/6})$ time (including the cost of reporting, without enumerating, the output canonical sets). See [1, Theorem 6.1] for more details.

The primal setup. With this procedure at hand, we go back to the primal structure, at each of the last four levels. As noted, the problem that we face there is a point enclosure problem, where the input consists of some N constant-complexity semi-algebraic regions in \mathbb{R}^6 of the form K_π , and the query is the point p_ℓ that represents ℓ , as defined earlier, and the task is to collect all the regions K_π that contain p_ℓ , as the disjoint union of a small number of precomputed canonical sets, or, at the last level, to determine whether p_ℓ is contained in any such region.

This problem has recently been studied in Agarwal et al. [2], using a multi-level polynomial partitioning technique, for the case where we allow maximum storage for the structure (that is, $O^*(N^6)$ in our case) and want the query time to be logarithmic. We next show that the structure can be modified so that its preprocessing stops “prematurely” when its overall storage attains some prescribed value, and each of the subproblems at the new leaves can be handled via the dual algorithm presented above.

The crucial technical tool in [2], on which their technique is based, is the following result. We give here a restricted specialized version that suffices for our purposes. (When applying this tool in d dimensions, the parameter 6 has to be replaced by d .)

► **Theorem 8** (A specialized version of Agarwal et al. [2, Corollary 4.8]). *Given a set Ψ of N constant-degree algebraic surfaces in \mathbb{R}^6 , and a parameter $0 < \delta < 1/6$, there are finite collections $\Omega_0, \dots, \Omega_6$ of semi-algebraic sets in \mathbb{R}^6 with the following properties.*

- *For each index i , each cell $\omega \in \Omega_i$ is a connected semi-algebraic set of constant complexity. The size $|\Omega_i|$ of Ω_i (the number of its sets) is a constant that depends on δ .*
- *For each index i and each $\omega \in \Omega_i$, at most $\frac{N}{4|\Omega_i|^{1/6-\delta}}$ surfaces from Ψ cross ω (intersect ω without fully containing it).*
- *The cells partition \mathbb{R}^6 , in the sense that $\mathbb{R}^6 = \bigsqcup_{i=0}^6 \bigsqcup_{\omega \in \Omega_i} \omega$, where \bigsqcup denotes disjoint union.*

The sets in $\Omega_0, \dots, \Omega_6$ can be computed in $O(n + m)$ expected time, where the constant of proportionality depends on δ , by a randomized algorithm. For each i and for every set $\omega \in \Omega_i$, the algorithm returns a semi-algebraic representation of ω , a reference point inside ω , and the subset of surfaces of Ψ that cross ω .

In our case, the surfaces of Ψ are the boundaries of the regions K_π . A straightforward enhancement of the algorithm of [2] also yields, for each i and each $\omega \in \Omega_i$, the set of regions K_π that fully contain ω , within the same asymptotic time bound.

We compute the partition of Theorem 8 and find, for each $\psi = \partial K_\pi \in \Psi$, the sets $\omega \in \Omega_i$, over all $i = 0, \dots, 6$, that ψ crosses, and those that are fully contained in K_π . For each i and $\omega \in \Omega_i$, let $\mathcal{K}_{i,\omega}$ (resp., $\mathcal{K}_{i,\omega}^0$) denote the set of tetrahedra $\Delta \in \mathcal{T}$ for which ∂K_π crosses ω (resp., K_π fully contains ω).

The overall size of the sets $\mathcal{K}_{i,\omega}^0$, over all i and $\omega \in \Omega_i$, is $O(N)$, with a constant that depends on δ (that is, on the sizes $|\Omega_i|$, which depend on δ).

For each i and ω we also have a recursive subproblem that involves the subset $\mathcal{K}_{i,\omega}$ of the tetrahedra Δ for which ∂K_π crosses ω . Putting $r_i := |\Omega_i|$, for $i = 0, \dots, 6$, we have, for each i and ω , $|\mathcal{K}_{i,\omega}| \leq \frac{N}{4r_i^{1/6-\delta}}$. We run the recursion, but not all the way through, as in [2].

Instead, we use the following storage allocation rule. We fix the storage that we are willing to allocate to the structure, and distribute it among the nodes of the recursion, as follows. To simplify the analysis, we distinguish between the storage itself, and the so-called *storage parameter* s , which is what we actually manage, but we have the property that the actual storage will always be $O^*(s)$.

Let s be the storage parameter that we allocate at the root of the structure. For each i and each set $\omega \in \Omega_i$, we allocate the storage parameter $s/(4|\Omega_i|)$ for ω . Hence, when we reach some set ω at a deeper level of recursion, say level j , the storage parameter allocated to ω is $\frac{s}{4^j |\Omega_{i_1}^{(1)}| \cdot |\Omega_{i_2}^{(2)}| \cdots |\Omega_{i_j}^{(j)}|}$, where $\Omega_{i_1}^{(1)}, \Omega_{i_2}^{(2)}, \dots, \Omega_{i_j}^{(j)}$, for indices $0 \leq i_1, i_2, \dots, i_j \leq 6$, are the partition families at the ancestors of ω in the recursion.

We stop the recursion when we reach nodes for which the allocated storage parameter is (roughly) equal to the number of tetrahedra at the node; a more precise statement of the termination rule is given shortly.

Put, for each set ω , $r_\omega := |\Omega_{i_1}^{(1)}| \cdot |\Omega_{i_2}^{(2)}| \cdots |\Omega_{i_j}^{(j)}|$, using the above notation for ω . The storage parameter allocated to ω is thus $s/(4^j r_\omega)$. Also, by Theorem 8, the number of tetrahedra Δ that participate in the subproblem at ω is at most

$$\frac{n}{4^j |\Omega_{i_1}^{(1)}|^{1/6-\delta} \cdot |\Omega_{i_2}^{(2)}|^{1/6-\delta} \cdots |\Omega_{i_j}^{(j)}|^{1/6-\delta}} = \frac{n}{4^j r_\omega^{1/6-\delta}},$$

and the stopping condition that we use is that $\frac{s}{4^j r_\omega} = \frac{n}{4^j r_\omega^{1/6-\delta}}$, or $r_\omega = (s/n)^{(6/5)/(1+6\delta/5)}$.

The size of a subproblem at a leaf is (using the $O^*(\cdot)$ notation to hide exponents that are proportional to δ and constants of proportionality that depend on δ)

$$n_\omega = \frac{n}{4^j r_\omega^{1/6-\delta}} = \frac{1}{4^j} O^* \left(\frac{n^{6/5}}{s^{1/5}} \right) = O^* \left(\frac{n^{6/5}}{s^{1/5}} \right).$$

In more detail, since all the parameters r_j in Theorem 8 are at least some sufficiently large constant that we can control, we can make the factor 4^j to be $O(s^\delta)$, for any $\delta > 0$ of our choice. To be more precise, the choice of δ determines how large the parameters r_j have to be taken to ensure that $4^j = O(s^\delta)$, and the choice of these parameters adds a constant factor to the query cost (incurred by the cost of locating, in brute force, the cells ω , at the various recursive levels, that contain p_ℓ), which depends on these parameters, and thus on δ .

At each leaf ω we pass to the dual structure reviewed above. It uses $O^*(n_\omega)$ storage and answers a query in time $O^*(n_\omega^{5/6}) = O^*(n/s^{1/6})$. To answer a query with a line ℓ in the combined structure, we search with its point p_ℓ in the primal substructure, in $O(\log n)$ time (with a constant of proportionality that depends on δ ; see below), to locate the leaf cell ω that contains p_ℓ (using the properties that (a) each recursive step involves a partitioning of constant size, and (b) the cells in the partition are pairwise disjoint). We then search with Q_ℓ in the dual structure at ω , which takes, as just noted, $O^*(n/s^{1/6})$ time. The overall cost of the query is therefore $O^*(n/s^{1/6})$.

As to the actual storage used by the structure, the allocation mechanism ensures that each level of the recursion uses storage that is at most $7/4$ times larger than the storage used in the previous level, because each node has seven child collections $\Omega_0, \dots, \Omega_6$, each of which is allocated an amount of storage $s/4$. Hence the overall storage used is $O((7/4)^j s)$, where j is the recursion depth. Arguing as in the query time analysis, we can make the factor $(7/4)^j$ to be $O(s^\delta)$, for any $\delta > 0$. That is, the overall storage used is $O(s^{1+\delta})$, or, in our notation, $O^*(s)$.

The above description of the structure applies to any single level among the four latter levels of the structure. The first two levels are considerably simpler and more efficient. The primal-dual approach is straightforward for halfspace range searching, and the parametric dimension is only four for the first two levels. The standard machinery (reviewed, e.g., in [1]) implies that, with s storage and N input tetrahedra, the cost of a query at each of these levels is $O^*(N/s^{1/4})$.

Putting everything together, and using standard arguments in the analysis of multi-level structures (see [1, Theorem 6.1] for details), the overall size of the six-level structure is $O^*(s)$, for any prescribed storage parameter s between n and n^6 , and a query takes $O^*(n/s^{1/6})$ time. That is, this finally concludes the proof of Theorem 1 for the case of intersection detection queries. Counting and reporting queries are handled similarly, with a similar analysis, exploiting the fact that the decomposition in Theorem 8 is into disjoint subsets, as is a similar decomposition used in the machinery of [11]. For reporting query, their cost involves an additional term $O(k)$, where k is the output size. \square

Remark. Our mechanism is in fact a special instantiation of the following general result, which is of independent interest, and which yields a trade-off bound for semi-algebraic range searching in any dimension d . That is, consider a general problem of this kind, that involves n points in \mathbb{R}^d , and aims to answer semi-algebraic range queries, where the ranges have constant complexity, and each range has d degrees of freedom (so the problem has a symmetric dual version). One can solve such a problem in time $O^*(n/s^{1/d})$ per query, using $O^*(s)$ space and preprocessing, where s is any parameter between n and n^d . These queries include detecting whether a query range contains any point from the input, counting the number of such points, or reporting them (with an additional term $O(k)$ in the query cost, where k is the output size). Using duality, we obtain the same performance bounds for point-enclosure queries, where the input consists on n constant-complexity semi-algebraic regions in \mathbb{R}^d , and the query is with a point p , where the goal is to detect, count or report containments of p in the input regions. The same asymptotic bound is obtained for simplex range searching [1], but our analysis shows that this bound corresponds to a much more general family of query ranges. The two extreme cases $s = n$ and $s = n^d$ have been treated in [11] and [2], respectively, but the tradeoff between these extreme cases has not been treated explicitly (for $d > 4$), as far as we can tell. As evidenced in the preceding analysis, this tradeoff is not as routine as one might think, because of the complicated nature of the partitioning used in Theorem 8 (as well as in [11, Theorem 1.1]). We summarize this result in the following corollary:

51:10 Intersection Searching Amid Tetrahedra in 4-Space

► **Theorem 9.** *Let P be a set of n points in \mathbb{R}^d , for any dimension d , and let Γ be a family of semi-algebraic ranges of constant complexity in \mathbb{R}^d , each of which has d degrees of freedom. Let $n \leq s \leq n^d$ be a prespecified storage parameter. Then one can preprocess P into a data structure of storage and preprocessing $O^*(s)$, such that a range-query, with a range $\gamma \in \Gamma$, can be answered in $O^*(n/s^{1/d})$ time. Such queries include detecting whether γ contains any point of P , counting the number of such points, and reporting them (with an additional $O(k)$ term in the latter case, where k is the number of these points). The same performance bounds apply to the dual point-enclosure case, where the input consists of n regions from Γ and the query is with a point $p \in \mathbb{R}^d$.*

Remarks.

- (i) Theorem 9 can be extended to the case where the number of degrees of freedom of the ranges is different from d , but the resulting performance bound has a more complicated expression, which is not spelled out in this work.
- (ii) We note that our technique can be extended to segment intersection detection queries amid a collection of n $(d-1)$ -simplices in any dimension d . In that case the structure has $d+2$ levels. The first two levels ensure that the endpoints of the query segment e lie on different sides of the hyperplane containing the input simplex Δ , and are implemented by halfspace range searching structures in \mathbb{R}^d . The last d levels ensure that the line containing e has positive orientation with respect to each of the $(d-2)$ -flats containing the facets of Δ , with suitable orientations of the line and the flats. Since lines and $(d-2)$ -flats in \mathbb{R}^d have $2d-2$ degrees of freedom, these levels are implemented using semi-algebraic range searching structures, where both primal and dual parts are in \mathbb{R}^{2d-2} . Hence the cost of the query at each of the last d levels dominates the overall cost, which is thus $O^*(n/s^{1/(2d-2)})$. The parameter s can vary between n and n^{2d-2} .
- (iii) A very similar mechanism, with the same performance bounds, handles the reverse situation, mentioned in the introduction, in which the input is a set of n segments in \mathbb{R}^4 , and the query is with a tetrahedron T , and the goal is to detect, count, or report intersections between T and the input segments. This property is stated in Theorem 7; we defer the relatively easy details to the full version.

3 Triangle-Triangle Intersection Queries in \mathbb{R}^4

Let Δ be a set of n triangles in \mathbb{R}^4 . We consider various triangle-triangle intersection problems, the simplest of which is just to detect whether the query triangle intersects any input triangle. Alternatively, we may want to count or to report all such intersections. For concreteness we consider only the detection problem in what follows, but, as in the previous section, the algorithm can be extended to also handle the other kinds of problems.

Similar to the preceding section, we use a multi-level data structure, where each level caters to one aspect of the condition that a triangle crosses another triangle. Specifically, let Δ_1 and Δ_2 be two triangles, and let π_1, π_2 be the respective 2-planes that contain them. Assuming general position, π_1 and π_2 always intersect at a single point ξ , and Δ_1 intersects Δ_2 if and only if ξ belongs to both triangles. As is easily verified, this latter condition is equivalent, with suitable orientations of π_1, π_2 , and of the lines supporting the edges of both triangles, to the conjunction of the following conditions:

- (i) π_1 is positively oriented with respect to each of the lines that support the edges of Δ_2 .
- (ii) π_2 is positively oriented with respect to each of the lines that support the edges of Δ_1 .

Conditions (i) and (ii) are the conjunction of a total of six sub-conditions, each of which tests the orientation of, say, the 2-plane π_1 with respect to the line supporting some specific edge of Δ_2 , or vice versa.

We can therefore apply a suitable variant of the same machinery of the preceding section, and obtain a proof of Theorem 3.

The batched bichromatic version. For the batched version of the triangle-triangle intersection problem, with m red triangles and n blue triangles (see Theorem 4), we choose the storage parameter s to be such that the cost of m queries with the red triangles is asymptotically roughly the same as the cost of preprocessing the blue triangles. That is, we set $s = mn/s^{1/6}$, or $s = m^{6/7}n^{6/7}$. For this choice to make sense, we need to ensure that $n \leq s \leq n^6$, or that $n^{1/6} \leq m \leq n^6$. When $m > n^6$ we only use the data structure of [2] and obtain the running time $O^*(m + n^6) = O^*(m)$, and when $m < n^{1/6}$ we only use the data structure of [11] and obtain the running time $O^*(mn^{5/6} + n) = O^*(n)$. Altogether we obtain the bound in Theorem 4.

4 Segment-Intersection amid Tetrahedra: An Improved Solution

In this section we present an improved algorithm for setup (i) of the paper, for a data structure of roughly quadratic size. Let \mathcal{T} be a collection of n tetrahedra in \mathbb{R}^4 . Our improved solution constructs a data structure that uses $O^*(n^2)$ storage (and expected preprocessing time), and answers a query in $O^*(n^{1/2})$ time. This is indeed a significant improvement over the standard algorithm in Section 2, in which, with storage $O^*(n^2)$, the query cost is $O^*(n^{2/3})$. With a suitable tradeoff, presented in the full version, the improvement can be extended for any storage parameter between n and n^6 , although it is most substantial when the storage is nearly quadratic; see Figure 1.

Assume, without loss of generality, that the query segment is bounded. The algorithm constructs a partitioning polynomial F in \mathbb{R}^4 of degree $O(D)$, for some large but constant parameter D , so that each cell of the partition is crossed by at most n/D^2 2-faces of the tetrahedra in \mathcal{T} and by a total of at most n/D tetrahedra. The existence of such a polynomial follows from Guth [8], and an expected linear-time algorithm for its construction (for constant D) is given in [2]. We classify each tetrahedron $\Delta \in \mathcal{T}$ as being *narrow* (resp., *wide*) with respect to a partition cell τ if a 2-face of Δ crosses τ (resp., Δ crosses τ but none of its 2-faces crosses τ). Let \mathcal{N}_τ (resp., \mathcal{W}_τ) denote the set of narrow (resp., wide) tetrahedra at τ .

There are two cases to consider in our analysis, depending on whether the query segment ρ is contained or not contained in the zero set $Z(F)$ of F . Each of these cases requires its own data structure. The latter case is an extension of the analysis in [7] (given there for the three-dimensional version of the problem), and the case where $\rho \subset Z(F)$ requires a different approach than that taken in [7] for handling queries on the zero set. Due to lack of space we only sketch the general framework; the details are given in the full version.

A sketch of the analysis. A query segment ρ that is not contained in $Z(F)$ crosses at most $O(D)$ cells of the partition. For each partition cell τ (an open connected component of $\mathbb{R}^4 \setminus Z(F)$) we construct an auxiliary data structure on the wide tetrahedra at τ , and preprocess the narrow tetrahedra at τ recursively. As we show below, the structure for the wide tetrahedra uses $S_0(n) = O^*(n^2)$ storage, and a query amid them takes $Q_0(n) = O^*(n^{1/2})$ time. We then output the wide tetrahedron returned by querying the auxiliary structure at τ , if such a tetrahedron exists. Otherwise, we return the tetrahedron produced by the

recursive call, if one exists. If no tetrahedron, wide or narrow, has been found, we proceed to the next cell τ' crossed by ρ , repeat the whole procedure at τ' , and keep doing this till we either find a tetrahedron hit by ρ or run out of cells, and then conclude that ρ does not hit any tetrahedron of \mathcal{T} .

The correctness of this procedure is clear (modulo that of the procedure for handling wide tetrahedra). Denote by $S(n)$ (resp., $Q(n)$) the maximum storage (resp., query time) required by the overall structure for n tetrahedra. Also denote by $S_1(n)$ (resp., $Q_1(n)$) the maximum storage (resp., query time) required for processing the input tetrahedra for intersection queries with segments contained in $Z(F)$, for any set of n tetrahedra in \mathbb{R}^4 . We then have, for a suitable absolute constant $c > 0$ (where the constant hidden in the $O_D(\cdot)$ notation depends on D),

$$\begin{aligned} S(n) &= O_D(S_0(n/D)) + S_1(n) + cD^4S(n/D^2) \\ Q(n) &= \max \{ O_D(Q_0(n/D)) + cDQ(n/D^2), Q_1(n) \}. \end{aligned}$$

We show, in the full version, that $S_1(n) = O_D^*(n^2)$ and $Q_1(n) = O_D^*(n^{1/2})$. Substituting these bounds, as well as the bounds for $S_0(n)$ and $Q_0(n)$, the solutions of these recurrences is $S(n) = O^*(n^2)$ and $Q(n) = O^*(n^{1/2})$. This establishes Theorem 2.

Handling the wide tetrahedra. Handling the wide tetrahedra is done via a secondary recursion, as follows. We choose some large constant parameter $r_0 \gg D$, and partition $\partial\tau$ into $O_D(1)$ $x_1x_2x_3$ -monotone strata (assuming a generic choice of the coordinate frame). This is fairly standard to do, see, e.g. [5]. We construct, for each stratum σ , a $(1/r_0)$ -cutting for the set of (constant-degree algebraic) 2-surfaces of intersection of σ with the wide tetrahedra in \mathcal{W}_τ . The cutting is constructed by projecting σ and the 2-surfaces that it contains onto the $x_1x_2x_3$ -subspace, constructing a $(1/r_0)$ -cutting, within that subspace, on the projected surfaces, and then lifting the resulting cutting back to σ . Using standard results on vertical decomposition in three dimensions (see, e.g., [13]) and the theory of cuttings [9], we obtain $O^*(r_0^3)$ cells of the cutting (referred to as (pseudo-)prisms, in accordance with the way in which the vertical-decomposition-based cutting is constructed), each of which is crossed by (intersects but not contained in) at most n/r_0 wide tetrahedra.

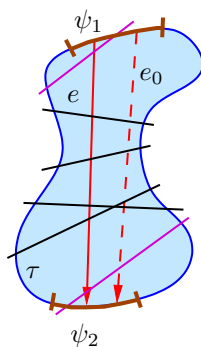
For each pair ψ_1, ψ_2 of prisms, we define S_{ψ_1, ψ_2} to be the set of all segments e so that e has an endpoint in ψ_1 and an endpoint in ψ_2 , and the relative interior of e is fully contained in τ . Clearly, S_{ψ_1, ψ_2} is a semi-algebraic set of constant complexity in a 6-dimensional parametric space,⁵ and we decompose it into its $O(1)$ connected components.

For each segment $e \in S_{\psi_1, \psi_2}$, let $\mathcal{T}(e)$ denote the set of all wide tetrahedra Δ of \mathcal{W}_τ that e crosses. We have the following crucial technical lemma, akin to Lemma 2.2 in [7]:

► **Lemma 10.** *Each connected component C of S_{ψ_1, ψ_2} can be associated with a fixed set \mathcal{T}_C of wide tetrahedra Δ of \mathcal{W}_τ , none of which crosses $\psi_1 \cup \psi_2$, so that, for each segment $e \in C$, $\mathcal{T}_C \subseteq \mathcal{T}(e)$, and each tetrahedron Δ in $\mathcal{T}(e) \setminus \mathcal{T}_C$ crosses either ψ_1 or ψ_2 .*

We illustrate the proof in Figure 10, and delegate the rest of the details to the full version of this paper.

⁵ Each segment is specified by its two endpoints; since they lie on $\partial\tau$, each has three degrees of freedom.



■ **Figure 2** The set \mathcal{T}_C (consisting of the tetrahedra depicted as black segments), and an illustration of the proof of Lemma 10: The tetrahedra that cross some fixed segment e_0 between ψ_1 and ψ_2 are the same tetrahedra that cross any other such segment e , except for those that cross ψ_1 or ψ_2 (like those depicted as magenta segments).

The analysis for wide tetrahedra. For each prism ψ , the *conflict list* K_ψ of ψ is the set of all wide tetrahedra that cross ψ . By construction, $|K_\psi| \leq n/r_0$. The same bound for crossing tetrahedra holds when ψ is lower-dimensional. If a lower-dimensional prism is contained in some tetrahedron there is no need to process ψ further, since any segment that meets ψ hits all these tetrahedra.

For each pair of prisms ψ_1, ψ_2 , we compute S_{ψ_1, ψ_2} and decompose it into its connected components. For each component C we compute the set \mathcal{T}_C of the wide tetrahedra, as in Lemma 10 (see the full version for details). This requires $O_D(r_0^6 n) = O_D(n)$ storage and computation time.

Let s be the storage parameter associated with the problem; we require that $n \leq s \leq n^3$. For each canonical set \mathcal{T}_C , we replace its tetrahedra by their supporting hyperplanes, and preprocess the resulting collection of hyperplanes for efficient segment intersection queries amid hyperplanes in \mathbb{R}^4 . Using the technique of Agarwal and Matoušek [3], this problem can be solved using $O^*(s)$ storage (and preprocessing), and a query takes $O(n \text{ polylog}(n)/s^{1/4}) = O^*(n/s^{1/4})$ time (see also [1]). Lemma 10 guarantees the correctness of this procedure (namely, that replacing each tetrahedron in \mathcal{T}_C by its supporting hyperplane does not cause any “false positive” answer).

We now process recursively each conflict list K_ψ , over all prisms ψ of the partition. Each recursive subproblem uses the same parameter r_0 , but the allocated storage parameter is now s/r_0^3 . We keep recursing until we reach conflict lists of size close to $n^{3/2}/s^{1/2}$. More precisely, after j levels of recursion, we get a total of at most $(c_0 r_0^3)^j = c_0^j r_0^{3j}$ subproblems, each involving at most n/r_0^j wide tetrahedra, for some constant c_0 that depend on D (but is considerably smaller than r_0).

We stop the recursion at the first level j^* at which $\frac{n}{r_0^{j^*}} \leq n^{3/2}/s^{1/2}$. As a result, we have $r_0^{j^*} = O(s^{1/2}/n^{1/2})$, and we get $c_0^{j^*} r_0^{3j^*} = O^*(s^{3/2}/n^{3/2})$ subproblems. Each of these subproblems involves at most $\frac{n}{r_0^{j^*}} = O^*\left(\frac{n^{3/2}}{s^{1/2}}\right)$ tetrahedra. Hence the overall size of the inputs, as well as of the canonical sets, at all the subproblems throughout the recursion, is $O^*\left(\frac{s^{3/2}}{n^{3/2}} \cdot \frac{n^{3/2}}{s^{1/2}}\right) = O^*(s)$. In particular, this is the asymptotic cost at the bottom level of the recursion.

51:14 Intersection Searching Amid Tetrahedra in 4-Space

As just described, at the bottom of the recursion, each subproblem contains at most $O^*(n^{3/2}/s^{1/2})$ wide tetrahedra, and we detect intersections with them by brute force. We thus obtain the following recurrence for the overall storage $S_0(N_W, s_W)$ for the structure constructed on N_W wide tetrahedra, where s_W denotes the storage parameter allocated to the structure (at the root $N_W = n, s_W = s$).

$$S_0(N_W, s_W) = \begin{cases} O_D^*(r_0^6 s_W) + c_0 r_0^3 S_0\left(\frac{N_W}{r_0}, \frac{s_W}{r_0^3}\right) & \text{for } N_W \geq \Theta^*(n^{3/2}/s^{1/2}), \\ O(N_W) & \text{for } N_W < \Theta^*(n^{3/2}/s^{1/2}). \end{cases}$$

Unfolding the recurrence up to the terminal level j^* , where $N_W = O^*(n^{3/2}/s^{1/2})$, the sum of the nonrecursive overhead terms, over all nodes at a fixed level j , is

$$c_0^j r_0^{3j} \cdot O^*\left(\frac{s_W}{r_0^{3j}}\right) = O^*(s_W).$$

Hence, starting the recurrence at $(N_W, s_W) = (n, s)$, the overall contribution of the overhead terms is $O^*(s)$. We showed above that this is also the asymptotic cost at the bottom of the recurrence. Therefore, the overall storage used by the data structure is $O^*(s)$. Using similar considerations, one can show that the overall expected preprocessing time is $O^*(s)$ as well, since the time obeys a similar asymptotic recurrence.

Answering a query. Given a query segment ρ , which is not contained in $Z(F)$, we find its $O(D)$ intersections with $Z(F)$, which decompose it into $O(D)$ segments, each fully contained in some partition cell. Moreover, except for the first and last segment, the endpoints of each of the other segments lie on the boundary of its cell. We process the segments in their order⁶ along ρ . Let e be the currently processed segment. If e is not the first or last segment, we find the prisms ψ_1, ψ_2 that contain its endpoints, and find the component C of S_{ψ_1, ψ_2} that contains e . If e is the first or last segment, we extend it backwards or forwards, respectively, till it meets the boundary of its cell, and call the resulting segment e' . We now compute for e' the corresponding set S_{ψ_1, ψ_2} and its component C that contains e' . Since D and r_0 are constants, all this takes constant time.

The query, on the wide tetrahedra at the present cell τ , performs a segment intersection detection query with e (or with e' when e is the first or last segment) in the set of hyperplanes containing the tetrahedra of \mathcal{T}_C , and, if no intersection is detected, continues recursively with \mathcal{T}_{ψ_1} and \mathcal{T}_{ψ_2} (at the bottom of recursion we apply a brute-force search). If no tetrahedron is found, in all the r_0 -recursive steps, we conclude that (the present subsegment of) ρ does not hit any wide tetrahedron within τ . Once again, the correctness of this procedure follows from Lemma 10.

The query time $Q_0(N_W, s_W)$ satisfies the recurrence

$$Q_0(N_W, s_W) = \begin{cases} O_D(1) + O^*\left(\frac{N_W}{s_W^{1/4}}\right) + 2Q\left(\frac{N_W}{r_0}, \frac{s_W}{r_0^3}\right) & \text{for } N_W \geq \Theta^*(n^{3/2}/s^{1/2}), \\ O(N_W) & \text{for } N_W < \Theta^*(n^{3/2}/s^{1/2}). \end{cases}$$

Unfolding the recurrence, the overall bound for the nonrecursive overhead terms, starting from $(N_W, s_W) = (n, s)$, is at most

$$O^*\left(\sum_{j \geq 0} \left(\frac{2}{r_0^{1/4}}\right)^j \cdot \frac{n}{s^{1/4}}\right) = O^*\left(\frac{n}{s^{1/4}}\right).$$

⁶ The order is immaterial for segment intersection detection queries, but is important for ray shooting.

Adding the cost at the 2^{j^*} subproblems at the bottom level j^* of the recursion, where the cost of each subproblem is at most $O^*(n^{3/2}/s^{1/2})$, we obtain the query time

$$Q_0(n, s) = O^* \left(\frac{n}{s^{1/4}} + \frac{n^{3/2}}{s^{1/2}} \right). \quad (3)$$

Therefore, for $s = n^2$ the query time is $O^*(n^{1/2})$. The bounds $S_0(n) := S_0(n, n^2) = O^*(n^2)$ and $Q_0(n) := Q_0(n, n^2) = O^*(n^{1/2})$ are the bounds promised earlier for the wide tetrahedra at a cell.

5 Output-Sensitive Construction of Arrangements of Tetrahedra and of Intersections of Polyhedra in \mathbb{R}^4

The results of Section 3 can be applied to construct the arrangement $\mathcal{A}(\mathcal{T})$ of a set \mathcal{T} of n tetrahedra in \mathbb{R}^4 in an output-sensitive manner. A complete discrete representation of $\mathcal{A}(\mathcal{T})$ requires, at the least, the collection of all faces, of all dimensions, of the arrangement, and their adjacency structure. Concretely, for each j -dimensional face φ , for $j = 0, 1, 2, 3$, we want the set of all $(j + 1)$ -dimensional faces that have φ on their boundary. Conversely, for each j -dimensional face φ , for $j = 1, 2, 3, 4$, we want the set of all $(j - 1)$ -dimensional faces that comprise $\partial\varphi$.

We begin by considering the task of computing all the nonempty intersections of pairs, triples, and quadruples of tetrahedra of \mathcal{T} . This will yield the set of vertices, and provide an infrastructure for computing the j -faces, for $j = 1, 2, 3$. Denote the number of these intersections as k_2 , k_3 , and k_4 , respectively. Note that we always have $k_4 \geq k_3 \geq k_2$.

To simplify the description we assume that the tetrahedra are in general position, although a suitable adaptation of the following machinery can handle degenerate cases too.

Reporting pairwise intersections. Two tetrahedra in general position in \mathbb{R}^4 intersect in a two-dimensional convex polygon of constant complexity, and it suffices to report one vertex of each nonempty polygon, in order to detect all intersecting pairs of tetrahedra. As is easily checked, such a vertex is either an intersection of an edge of one tetrahedron with the other tetrahedron, or an intersection of two 2-faces (triangles), one from each tetrahedron.

Reporting vertices of the first kind (edge-tetrahedron intersections) can be done using the machinery in Theorem 1, whose details are provided in Section 2, which takes $O^*(n^{12/7} + k_2)$ time.⁷ Reporting vertices of the second kind (triangle-triangle intersections) is done using the machinery in Section 3, which also takes $O^*(n^{12/7} + k_2)$ time.

Reporting triple and quadruple intersections. We iterate over the input tetrahedra. For each fixed tetrahedron T_0 , the previous step provides us with all the other tetrahedra that intersect T_0 . Denote their number as k_{T_0} , and observe that $\sum_{T_0} k_{T_0} = 2k_2$. We form the nonempty intersections $T_0 \cap T$, and triangulate each of them. We obtain a collection of $O(k_{T_0})$ triangles, all contained in $(T_0$ and therefore also in) the hyperplane h_{T_0} supporting T_0 .

We have thus reduced our problem to that of reporting all pairwise and triple intersections in a set of $m = O(k_{T_0})$ triangles in \mathbb{R}^3 . This can be solved using the algorithm in [7], by a procedure that runs in $O^*(m^{3/2} + \ell_{T_0})$ time, where ℓ_{T_0} is the number of triple intersections of the triangles. Note that $\sum_{T_0} \ell_{T_0} = O(k_4)$.

⁷ Although this part can be performed faster, as described in Section 4, we use the standard solution, since we do not have a similar improvement for the construction of vertices of the second kind.

Adding up this cost over all tetrahedra T_0 , the overall running time is

$$O^* \left(\sum_T k_T^{3/2} + k_4 \right) = O^* \left(n^{1/2} \sum_T k_T + k_4 \right) = O^*(n^{1/2}k_2 + k_4).$$

Constructing the arrangement. For each tetrahedron T_0 , it is fairly routine to obtain, from the information collected so far, the full three-dimensional arrangement within T_0 , using standard techniques in three dimensions; we omit here these standard details. This gives us all the j -faces of the four-dimensional arrangement \mathcal{A} , for $j = 0, 1, 2, 3$, and their adjacency information. The local adjacency information in \mathbb{R}^4 is also available from this data. By local adjacency we mean the adjacency between a j -face and the j' -faces on its boundary, for $j' < j$, over all such pairs of faces. For completion we need to identify disconnected pieces of the boundary of each four-dimensional cell, and record their adjacency to that cell. This can be done by x_4 -vertical ray shooting from the x_4 -highest point of each connected three-dimensional complex of faces. This calls for performing $O(n)$ x_4 -vertical ray shooting queries in a set of n tetrahedra in \mathbb{R}^4 , which can be done using the machinery presented in Theorem 1, or by an even simpler mechanism (since all the rays are vertical).

We have thus established the bound stated in Theorem 6.

Output-sensitive construction of the intersection of polyhedra in \mathbb{R}^4 . As another application, consider the problem where we have two not necessarily convex polyhedra R and B in \mathbb{R}^4 , whose boundaries consist of, or can be triangulated into $O(n)$ faces of all dimensions, which are segments, triangles, and tetrahedra. The goal is to construct their intersection $R \cap B$ in an output-sensitive manner; a similar application has been shown in [7] for the three-dimensional problem. We note that computing the union $B \cup R$ can be done using a very similar approach, within the same asymptotic time bound.

In order to compute $R \cap B$, we first apply the above algorithm to construct, in an output-sensitive manner, the arrangement $\mathcal{A}(R \cup B)$ of the two polyhedra R and B (specifically, we build the arrangement of the tetrahedra comprising the boundaries of B and R). We then label each cell (of any dimension) of $\mathcal{A}(R \cup B)$ with the appropriate Boolean operation, that is, whether it either lies in $R \setminus B$, $B \setminus R$, $B \cap R$, or in the complement of $B \cup R$. Collecting all the cells of the desired kind (e.g., those in $B \cap R$), and computing the adjacency relation between them, we obtain a suitable representation of the intersection. This establishes the bound stated in Theorem 6(ii).

We comment that extending the analysis to the intersection of more than two (albeit, still a constant number of) input polyhedra can also be done, following the same machinery as in the construction of an arrangement of tetrahedra, as presented above. It is easy to verify that in this case we obtain the same asymptotic bound stated in Theorem 6(ii).

References

- 1 P. K. Agarwal. Simplex range searching and its variants: A review. In *Journey through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 1–30. Springer Verlag, Berlin-Heidelberg, 2017.
- 2 P. K. Agarwal, B. Aronov, E. Ezra, and J. Zahl. An efficient algorithm for generalized polynomial partitioning and its applications. *SIAM J. Comput.*, 50:760–787, 2021. Also in *Proc. Sympos. on Computational Geometry (SoCG)*, 2019, 5:1–5:14. Also in [arXiv:1812.10269](https://arxiv.org/abs/1812.10269).
- 3 P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22:794–806, 1993.

- 4 P. K. Agarwal, J. Matoušek, and M. Sharir. On range searching with semialgebraic sets II. *SIAM J. Comput.*, 42:2039–2062, 2013. Also in [arXiv:1208.3384](#).
- 5 S. Basu, R. Pollcák, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer-Verlag, Berlin–Heidelberg, 2nd edition, 2006.
- 6 J. Canny. Collision detection for moving polyhedra. *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, 8:200–209, 1986.
- 7 E. Ezra and M. Sharir. On ray shooting for triangles in 3-space and related problems. *SIAM J. Comput.*, to appear. Also in Proc. 37th Sympos. on Computational Geometry, 2021, 34:1–34:15, and in [arXiv:2102.07310](#).
- 8 L. Guth. Polynomial partitioning for a set of varieties. *Math. Proc. Camb. Phil. Soc.*, 159:459–469, 2015. Also in [arXiv:1410.8871](#).
- 9 D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.
- 10 M. C. Lin, D. Manocha, and Y. J. Kim. Collision and proximity queries. In *Handbook on Discrete and Computational Geometry*, chapter 39, pages 1029–1056. CRC Press, Boca Raton, Florida, 3rd edition, 2017.
- 11 J. Matoušek and Z. Patáková. Multilevel polynomial partitions and simplified range searching. *Discrete Comput. Geom.*, 54:22–41, 2015.
- 12 E. Schömer and Ch. Thiel. Efficient collision detection for moving polyhedra. In *Proc. 11th Sympos. on Computational Geometry*, pages 51–60, 1995.
- 13 M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, Cambridge-New York-Melbourne, 1995.

Submodular Maximization Subject to Matroid Intersection on the Fly

Moran Feldman 

University of Haifa, Israel

Ashkan Norouzi-Fard 

Google Research, Zürich, Switzerland

Ola Svensson 

EPFL, Lausanne, Switzerland

Rico Zenklusen  

ETH Zürich, Switzerland

Abstract

Despite a surge of interest in submodular maximization in the data stream model, there remain significant gaps in our knowledge about what can be achieved in this setting, especially when dealing with multiple constraints. In this work, we nearly close several basic gaps in submodular maximization subject to k matroid constraints in the data stream model. We present a new hardness result showing that super polynomial memory in k is needed to obtain an $o(k/\log k)$ -approximation. This implies near optimality of prior algorithms. For the same setting, we show that one can nevertheless obtain a constant-factor approximation by maintaining a set of elements whose size is independent of the stream size. Finally, for bipartite matching constraints, a well-known special case of matroid intersection, we present a new technique to obtain hardness bounds that are significantly stronger than those obtained with prior approaches. Prior results left it open whether a 2-approximation may exist in this setting, and only a complexity-theoretic hardness of 1.91 was known. We prove an unconditional hardness of 2.69.

2012 ACM Subject Classification Theory of computation \rightarrow Communication complexity

Keywords and phrases Submodular Maximization, Matroid Intersection, Streaming Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.52

Related Version *Full Version:* <https://arxiv.org/pdf/2204.05154.pdf>

Funding *Moran Feldman:* Research supported in part by the Israel Science Foundation (ISF) grant no. 459/20.



Ola Svensson: Research supported by the Swiss National Science Foundation project 200021-184656 “Randomness in Problem Instances and Randomized Algorithms.”



Rico Zenklusen: Research supported in part by Swiss National Science Foundation grant number 200021_184622. 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 817750).



1 Introduction

A set function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}$ over a ground set \mathcal{N} is *submodular* if

$$f(u \mid A) \geq f(u \mid B) \quad \text{for all } A \subseteq B \subseteq \mathcal{N} \text{ and } u \in \mathcal{N} \setminus B,$$

where, for a subset $S \subseteq \mathcal{N}$ and an element $u \in \mathcal{N}$, we denote by $f(u \mid S) := f(S \cup \{u\}) - f(S)$ the marginal contribution of u with respect to S . We say that f is *monotone* if $f(A) \leq f(B)$ for any $A \subseteq B \subseteq \mathcal{N}$.



© Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 52; pp. 52:1–52:14



Leibniz International Proceedings in Informatics

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

The definition of submodular functions captures the natural property of diminishing returns, and the study of these functions has a rich history in optimization with numerous applications (see, e.g., the book [28]). Already in 1978, Nemhauser, Wolsey, and Fisher showed that a natural greedy algorithm achieves a tight approximation guarantee of $\frac{e}{e-1}$ for selecting the most valuable subset $S \subseteq \mathcal{N}$ of cardinality at most ρ (see [26] for the algorithm’s analysis and [8, 25] for matching hardness results). Since then, significant work has been devoted to extending their result to more general constraints.

A natural generalization of a cardinality constraint is the class of matroid constraints. While matroid constraints are much more expressive than cardinality constraints, both constraints often enjoy the same (or similar) algorithmic guarantees. Indeed, for the problem of maximizing a monotone submodular function subject to a single matroid constraint, Călinescu, Chekuri, Pál, and Vondrák [3] developed the continuous greedy method, and showed that it extends the $\frac{e}{e-1}$ -approximation guarantee to this more general setting. Moreover, for the maximization of a monotone submodular function subject to $k \geq 2$ matroid constraints, there is a $(k+1)$ -approximation guarantee by Fisher, Nemhauser and, Wolsey [11], which was improved to $k + \varepsilon$ by Lee, Sviridenko, and Vondrák [22] when the number k of matroid constraints is considered to be a constant.

While these algorithms are efficient in the traditional sense, i.e., they run in polynomial time in the offline “RAM” model, recent applications in data science and machine learning [21] with very large-scale problem instances have motivated the need for very space-efficient algorithms. In particular, it is interesting to study algorithms whose memory footprint is *independent of the ground set size*.¹ The task of designing such algorithms for (monotone) submodular function maximization has become a very active research area, especially in the context of the popular data stream computational model. Recent progress has resulted in a tight understanding of data stream algorithms² for maximizing monotone submodular functions with a single cardinality constraint: one can obtain a 2-approximation for this problem using a simple “threshold”-based algorithm that requires only $\tilde{O}(\rho)$ memory [2, 19], where ρ is the maximum number of elements allowed in the solution, and this is essentially optimal unless one is willing to have a space complexity that is linear in the size of the ground set [9]. However, our understanding of data stream algorithms for more general constraint families is currently much more limited. Closing this gap for natural settings of multiple matroid constraints is the motivation for our work.

Formally, we term the problem that we study **Submodular Maximization subject to k Matroid Constraints (SMkM)**. In this problem, we are given $k \geq 2$ matroids $M_1 = (\mathcal{N}, \mathcal{I}_1), M_2 = (\mathcal{N}, \mathcal{I}_2), \dots, M_k = (\mathcal{N}, \mathcal{I}_k)$ sharing a common ground set \mathcal{N} , and a non-negative submodular function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}_{\geq 0}$.³ Our goal is to find a *common independent set* $S \subseteq \mathcal{N}$ (i.e., S is independent in all the matroids) that maximizes $f(S)$. In the data stream

¹ Technically, a logarithmic dependence on the ground set size is unavoidable because, at the very least, the algorithm has to store the indices of the elements in its solution. However, we wish to have a space complexity whose dependence on the ground set size is limited to this unavoidable logarithmic dependence.

² Data stream algorithms are sometimes called “streaming algorithms”; however, in this paper we reserve the term “streaming algorithms” to data streaming algorithms whose space complexity is poly-logarithmic in the natural parameters of the problem.

³ We recall that a matroid $M = (\mathcal{N}, \mathcal{I})$ is a tuple consisting of a finite ground set \mathcal{N} and a nonempty family $\mathcal{I} \subseteq 2^{\mathcal{N}}$ of subsets thereof fulfilling (i) if $I \in \mathcal{I}$ and $J \subseteq I$, then $J \in \mathcal{I}$, and (ii) if $I, J \in \mathcal{I}$ with $|I| < |J|$, then there is an element $e \in J \setminus I$ such that $I \cup \{e\} \in \mathcal{I}$. Moreover, a function $f: 2^{\mathcal{N}} \rightarrow \mathbb{R}_{\geq 0}$, where \mathcal{N} is a finite set, is *submodular* if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. This is well-known to be equivalent to the *diminishing returns* property, which states that for $A \subseteq B \subseteq \mathcal{N}$ and $e \in \mathcal{N} \setminus B$, we have $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$.

version of this problem, the elements of the ground set \mathcal{N} appear one by one on a stream, and the algorithm should make a single pass over this stream and construct its output set S . (Some papers allow also algorithms that can do a few sequential passes over the stream; however, we consider the more practical and fundamental single-pass setting.)

The above high-level description of **SMkM** hides some important technical details regarding the way in which the objective function f and the matroid constraints are accessed. In the literature about matroids, it is customary to assume that the access of an algorithm to a matroid is done via an *independence oracle* that, given a set $S \subseteq \mathcal{N}$, indicates whether S is independent in the matroid. This notion can be extended to the intersection of k matroids in two natural ways: (i) having a single *common independence oracle*, which indicates whether S is independent in this intersection (i.e., in all the matroids), or (ii) having k independence oracles, one per matroid. Let us first consider the model with weaker access to the matroids, i.e., the one with a common independence oracle. This model already allows the implementation of a simple algorithm that greedily adds to its solution every element that does not violate feasibility. This natural greedy algorithm is a k -approximation for the special case of **SMkM** in which f is simply the cardinality function (i.e., $f(S) = |S|$) [17, 20] using a space complexity of $\tilde{O}(\rho)$, where ρ is the *common rank* of the k matroids, i.e., the cardinality of a maximum cardinality common independent set in the k matroids of the **SMkM** instance.⁴ We can show that this simple algorithm is almost best possible when access to the matroid is restricted to calls to a common independence oracle, unless one is willing to have a space complexity that is linear in $n := |\mathcal{N}|$.

► **Theorem 1.** *A data stream algorithm for **SMkM**, whose only access to the matroids is via the common independence oracle, and with expected approximation ratio $k - \varepsilon$ (for some $\varepsilon \in [0, k - 1)$), must use $\Omega(\varepsilon^n / k^5 \log k)$ memory. This holds even when the task is to find a maximum size common independent set in k partition matroids, and the common rank of these matroids is k .*

The proof of Theorem 1 is based on carefully defining k matroids such that stream prefixes lead to restricted matroids with many indistinguishable elements. This allows for hiding a large optimal solution. See the full version for details.

Given this inapproximability result, we turn our focus to the model in which we have access to a separate independence oracle for every matroid. A $4k$ -approximation algorithm with space complexity $\tilde{O}(\rho)$ was given for **SMkM** in this model by Chakrabarti and Kale [5] when the objective function f is guaranteed to be monotone, and $O(k)$ -approximation algorithms with similar space complexities were later obtained for the general case by Chekuri et al. [6] and Feldman et al. [10]. Our first main result shows that improving over the approximation guarantees of these algorithms by more than a logarithmic factor requires super polynomial space in k . Specifically, we prove the following theorem.

► **Theorem 2.** *Any data stream algorithm for **SMkM** that finds an α -approximate solution with probability at least $2/3$ uses memory at least $\Omega(e^{k/(8\alpha)} / k^2)$ assuming $\alpha \leq k / (32 \ln k)$. This holds even when the task is to find a maximum size common independent set in k partition matroids, and the common rank of these matroids is $O(\alpha) = O(k / \log k)$.*

The technique to prove Theorem 2 is discussed in Section 3. Interestingly, this technique also implies that any (even preemptive) online algorithm for **SMkM** must also have an approximation ratio of $\Omega(k / \log k)$. We remark that this lower bound is asymptotically the same as

⁴ For general **SMkM**, the state-of-the-art algorithm with a space complexity of $\tilde{O}(\rho)$ obtains a slightly worse approximation ratio of $O(k \log k)$ with a common independence oracle [12]. Moreover, this algorithm is applicable even to the more general class of k -extendible constraints.

the well-known approximation hardness for k -dimensional matching [13], which is a special case of the intersection of k matroids. However, note that this hardness does not carry over to our setting as our model has no restriction on computational power but only on memory.

In light of Theorem 2, it is arguably surprising that one can get essentially a 2-approximation for SMkM with space complexity independent of n .⁵

► **Theorem 3.** *For every $\varepsilon \in (0, 1/7)$, there exists a $(2 + O(\varepsilon))$ -approximation data stream algorithm for SMkM with space complexity $\text{poly}(k^{\rho^2 k}, \varepsilon^{-\rho})$.*

For monotone objective functions, Theorem 3 is based on merging ideas that appeared in recent papers by Huang et al. [14] and Huang and Ward [16] (see the full version for details). However, to obtain the same guarantee for non-monotone functions requires an interesting novel guessing scheme. Moreover, this theorem cannot be improved by much. The exponential dependence on k is necessary by Theorem 2, and the approximation ratio cannot be improved, even for a cardinality constraint, without using a linear in n memory because of the inapproximability result of [9]. It is open (even for a single partition matroid constraint) whether the exponential dependence on ρ in Theorem 3 is necessary.

Up to this point, our inapproximability results concentrated on the asymptotic approximation ratio obtainable as a function of k , which is more relevant for large values of k . Small values of k have also been considered extensively in the literature. For example, as aforementioned, the approximation ratio that can be obtained by a data streaming algorithm for a cardinality constraint, which is a special case of SMkM with $k = 1$, was the subject of a long line of research [1, 2, 9, 14, 19, 24] and is now essentially settled. Maximizing a monotone submodular function subject to a bipartite matching constraint is another important special case of SMkM, this time for $k = 2$.

Since ρ is usually polynomially related to n in bipartite matching constraints, the class of algorithms considered interesting for the last problem is more restricted than for general SMkM. Specifically, people are interested in algorithms that use $\tilde{O}(\rho)$ memory. That is, the algorithm does not use more memory (up to logarithmic factors) than what is required to simply store a solution. Such algorithms are known as semi-streaming algorithms. Recently, Levin and Wajc [23] described a semi-streaming algorithm for maximizing a monotone submodular function subject to a bipartite matching constraint which improves over the state-of-the-art for general SMkM with a monotone objective function. They also proved, conditioned on some complexity-theoretic assumption, a lower bound of 1.914 on the approximation ratio that can be obtained by a semi-streaming algorithm for the problem. Our final result improves over this upper bound and is independent of any complexity-theoretic assumption, but does assume that the graph can contain parallel edges (which are distinct elements from the point of view of the submodular objective function); the hardness of [23] applies even when this is not the case.

► **Theorem 4.** *No semi-streaming algorithm can obtain, with probability at least $2/3$, an approximation ratio of 2.692 for maximizing a non-negative monotone submodular function subject to a bipartite matching constraint.*

The last result is obtained by combining known hardness results for semi-streaming algorithms for the maximum cardinality matching problem [18] and submodular function maximization subject to a cardinality constraint [9] in a non-trivial way so that the obtained

⁵ For simplicity, the space complexity stated in Theorem 3 assumes that every element of the ground set can be stored in $O(1)$ space. Without this assumption, we get the unavoidable logarithmic dependence of the space complexity on n . Similarly, we also make the standard assumption that the value of $f(S)$ can be stored in constant space for every set $S \subseteq \mathcal{N}$.

hardness is stronger than what is known for any one of the two problems individually. Moreover, the reduction is general, and another consequence of it is the following: any semi-streaming algorithm for maximizing a monotone submodular function subject to a bipartite matching constraint that has an approximation guarantee better than 3 would yield an improved semi-streaming algorithm for the maximum cardinality bipartite matching problem, which is a longstanding notorious open problem. We refer to reader to the full version for further detail.

2 Preliminaries

In this section we give some additional technical details that are necessary for proving the results stated in Section 1. In Section 2.1 we discuss in more detail the oracles used to access the objective function and constraint matroids; and in Section 2.2 we present a known hard problem from which we often reduce to prove our inapproximability results.

2.1 More details about the access oracles

As mentioned above, the matroid constraints are accessed via either a common independence oracle or k distinct independence oracles, one per matroid. We also need to specify the method used to access the objective function f . In the submodular optimization literature, submodular objective functions such as f are usually accessed via a *value oracle* that, given a set $S \subseteq \mathcal{N}$, returns $f(S)$. In the context of data stream and online algorithms, it is important that the independence and value oracles do not leak information in a way that contradicts our expectations from such algorithms. Accordingly, our algorithms query the oracles only on sets of elements that are explicitly stored in their memory.

The above information leakage issue often makes proving inapproximability results more complicated because such results have to formalize in some way the types of queries that are allowed (i.e., queries that are not considered “leaky”). All our inapproximability results apply to the model used by our algorithmic results; namely, when the algorithm is allowed to query the oracles only on sets of elements that are explicitly stored in its memory – see [15] for a formal statement of this natural model. However, we strive to weaken this assumption, and prove most of our inapproximability results even for algorithms that enjoy a less limited access to the oracles. For example, the proof of Theorem 2 manages to avoid this issue completely using the following technique. The algorithm is given upfront a “super ground set” and fully known objective function and matroids over this super ground set. The real ground set is then chosen as some subset of this super ground set, and only the elements of this real ground set appear in the input stream of the algorithm. Since the challenge that the algorithm has to overcome in this case is to remember which elements belong to the real ground set, the oracles cannot leak important information to the algorithm, and therefore, we allow the algorithm unrestricted access to them.

The situation for Theorem 1 is a bit more involved because of the following observation. If the algorithm is allowed unrestricted access to the common independence oracle, then it can construct k matroids M_1, M_2, \dots, M_k that are consistent with this oracle, which makes the distinction between a single common independence oracle and k independence oracles mute. Therefore, some restriction on the access to the common independence oracle must be used. The (arguably) simplest and most natural restriction of this kind is to allow the algorithm to query the common independence oracle only on subsets that do not include any elements that did not appear in the stream so far; and it turns out that this simple restriction suffices for the proof of Theorem 1 to go through.

It remains to consider our last inapproximability result, namely Theorem 4. Here the elements of the ground set are edges, and the algorithm is given each edge in the form of its two endpoints. Therefore, there is no need for independence oracles. (Formally, this situation is equivalent to the case mentioned above in which there is a “super ground set” that is given upfront to the algorithm, and only part of this super ground set appears in the stream.) Unfortunately, preventing information leakage via the value oracle is more involved. For simplicity, the proof that we give in the full version assumes the same model that we use in our algorithmic results. However, our proof can be extended also to algorithms with a more powerful way to access the objective function f , such as the p -players model described in [9].

2.2 The $\text{CHAIN}_p(n)$ problem

Many of our inapproximability results use reductions to a (hard) problem named $\text{CHAIN}_p(n)$, introduced by Cormode, Dark, and Konrad [7], which is closely related to the Pointer Jumping problem (see [4]). In this problem, there are p players P_1, \dots, P_p . For every $1 \leq i < p$, player P_i is given a bit string $x^i \in \{0, 1\}^n$ of length n , and, for every $2 \leq i \leq p$, player P_i (also) has as input an index $t^i \in \{1, 2, \dots, n\}$. (Note that the convention in this terminology is that the superscript of a string/index indicates the player receiving it.) Furthermore, it is promised that either $x_{t^{i+1}}^i = 0$ for all $1 \leq i < p$ or $x_{t^{i+1}}^i = 1$ for all these i values. We refer to these cases as the 0-case and 1-case, respectively. The objective of the players in $\text{CHAIN}_p(n)$ is to decide whether the input instance belongs to the 0-case or the 1-case. The first player, based on the input bit string x^1 , sends a message M^1 to the second player. Any player $2 \leq i < p$, based on the message it receives from the previous player (i.e., M^{i-1}), the input bit string x^i and index t^i , sends message M^i to the next player. The last player, based on M^{p-1} and t^p , decides if we are in the 0-case or 1-case. Each player has unbounded computational power and can use any (potentially randomized) algorithm. We refer to the collections of the algorithms used by all the players as a protocol. The success probability of a protocol is the probability that its decision is correct, and the communication complexity of a protocol is the size of the maximum message sent (i.e., maximum size of M^1, \dots, M^{p-1}). In [9], the following lower bound was shown for the $\text{CHAIN}_p(n)$ problem, which is very similar to the lower bounds previously proved by [7].

► **Theorem 5** (Theorem 3.3 in [9]). *For any positive integers n and $p \geq 2$, any (potentially randomized) protocol for $\text{CHAIN}_p(n)$ with success probability of at least $2/3$ must have a communication complexity of at least $n/36p^2$. Furthermore, this holds even when instances are drawn from a known distribution $D(p, n)$.*

The distribution $D(p, n)$ referred to by Theorem 5 is simply the uniform distribution over all 0-case and 1-case instances (see the definition of D^p in Appendix C of [9]).

3 Inapproximability for Multiple Independence Oracles

In this section, we prove Theorem 2, which gives a strong inapproximability result for data stream algorithms as a function of the number k of matroids, even in the case when the objective function f is a linear function (unlike in the previous section, here we allow access to the independence oracles of the individual matroids).

► **Theorem 2.** *Any data stream algorithm for SMkM that finds an α -approximate solution with probability at least $2/3$ uses memory at least $\Omega(e^{k/(8\alpha)}/k^2)$ assuming $\alpha \leq k/(32 \ln k)$. This holds even when the task is to find a maximum size common independent set in k partition matroids, and the common rank of these matroids is $O(\alpha) = O(k/\log k)$.*

Note that the above result implies that (i) any data stream algorithm with an approximation guarantee $o(k/\log k)$ requires super polynomial memory in k , and (ii) any data stream algorithm with constant approximation guarantee requires exponential memory in k .

The techniques we use also readily imply hardness for the (preemptive) online version of this problem. In this version, the elements of the ground set \mathcal{N} arrive online, and upon receiving each element the algorithm has to decide either to add this element to the solution it maintains, or to reject the element. If the algorithm accepts an element to its solution, it may remove this element from the solution at a later point; however, a decision to reject an element (or remove it from the solution at a later time) is irrevocable. The algorithm is also required to keep its solution feasible at all times. We have the following hardness in this model.

► **Theorem 6.** *For $k \geq 2$, the competitive ratio of any online algorithm for $SMkM$ against an oblivious adversary is at least $\frac{k}{81 \ln k}$. This holds even when the task is to find a maximum size common independent set in k partition matroids, and the common rank of these matroids is $O(k/\log k)$.*

The key building block for both (streaming and online) hardness results is a “hard” distribution of instances described in Section 3.1. This distribution is then used to prove Theorems 2 and 6 in Sections 3.2 and 3.3, respectively, in a similar way to the proof of Theorem 1.

3.1 Description of hard distribution

Let p be a non-negative integer parameter of the construction. Our instances are subsets of the set $\mathcal{N} = [p]^k$, where we allow multiple elements with the same coordinates (i.e., “multi-subsets”). A set $S \subseteq \mathcal{N}$ is independent in the matroid M_i (for any integer $i \in [k]$) if and only if no two elements of S share the same value in coordinate number i (in other words, $u_i \neq v_i$ for every two distinct elements $u, v \in S$). One can observe that this definition makes M_i a partition matroid. We recall that $S \subseteq \mathcal{N}$ is a *common independent set* if it is independent in all matroids; otherwise, we will refer to it as *dependent*. Note that the common rank of the k matroids M_1, \dots, M_k is p .

In Algorithm 1 we describe a procedure for sampling $p-1$ many subsets $S_1, S_2, \dots, S_{p-1} \subseteq [p]^k$ and $p-1$ “hidden” optimal elements $o^{(1)} \in S_1, o^{(2)} \in S_2, \dots, o^{(p-1)} \in S_{p-1}$. In every iteration $r = 1, \dots, p-1$, the algorithm first forms S_r by sampling m elements independently and uniformly from those elements that form a common independent set with $\{o_1, \dots, o_{r-1}\}$. That is, S_r contains m uniformly random samples with replacements from

$$\{u \in \mathcal{N} \mid \forall 1 \leq i < r, 1 \leq j \leq k \ o_j^{(i)} \neq u_j\} .$$

Then, *after* the selection of S_r , the algorithm samples $o^{(r)}$ uniformly at random among the m elements in S_r .

We remark that the algorithm with small probability may sample the same element more than once when forming the set S_r . When this happens, we consider these samples to be unique elements on the stream (that are dependent). This allows us to simplify the notation in the following as each set S_r is now guaranteed to contain exactly m elements. Formally, this corresponds to extending the ground set \mathcal{N} by making m copies u^1, \dots, u^m of each element $u \in \mathcal{N}$, and whenever an element u is sampled i times, we include the copies u^1, \dots, u^i .

Algorithm 1 HARD INSTANCE GENERATION (p, m) .

- 1: **for** $r = 1$ **to** $p - 1$ **do**
 - 2: Obtain S_r by sampling m elements uniformly and with replacement from

$$\{u \in \mathcal{N} \mid \forall_{1 \leq i < r, 1 \leq j \leq k} o_j^{(i)} \neq u_j\} .$$
 - 3: Select $o^{(r)}$ from S_r uniformly at random.
 - 4: Output S_1, \dots, S_{p-1} and $o^{(1)}, \dots, o^{(p-1)}$.
-

We refer to Algorithm 1 as “HARD INSTANCE GENERATION” as it is the basic building block of our hardness results in both the online and streaming models. More specifically, our hardness result for the online model is based on the (random) stream obtained by first feeding the elements in S_1 (in any order), then S_2 (in any order), and so on until S_{p-1} is fed. The intuition is that when the algorithm has only seen the elements in S_1, \dots, S_i , it has no information about the selection of $o^{(i)}$ and so any online algorithm is unlikely to have saved the element $o^{(i)}$. In addition, while $\{o^{(1)}, \dots, o^{(p-1)}\}$ is a common independent set by construction, we prove (see Lemma 8 below) that any other two elements are likely to be dependent. This creates the “gap” between the values of the solution $\{o^{(1)}, o^{(2)}, \dots, o^{(p-1)}\}$ and a solution with any other elements, which in turn yields the desired hardness result. For our hardness in the data stream model, we forward a subset of the above-mentioned stream, and the difficulty for a low-space streaming algorithm is to “remember” whether the special elements $o^{(1)}, o^{(2)}, \dots, o^{(p-1)}$ appeared in the stream. This is formalized in the next sections.

We complete this section by proving that, with good probability, any large solution must contain the hidden elements $o^{(1)}, \dots, o^{(p-1)}$.

► **Definition 7.** *We say that the output of Algorithm 1 is successful if any two elements $e, f \in S_1 \cup S_2 \cup \dots \cup S_{p-1} \setminus \{o_1, \dots, o_{p-1}\}$ are dependent, i.e., there is a coordinate $i \in [k]$ such that $e_i = f_i$.*

► **Lemma 8.** *The output of Algorithm 1 is successful with probability at least $1 - \binom{pm}{2} e^{-k/p}$.*

Proof. Consider two elements $u \in S_{r_1} \setminus \{o_{r_1}\}$ and $v \in S_{r_2}$ with $r_1 \leq r_2$. As $u \neq o_{r_1}$, each coordinate of u equals that of v with probability at least $1/(p - r_1 + 1) \geq 1/p$. Now, as there are k coordinates, and each coordinate of u is sampled independently at random,

$$\Pr[\{u, v\} \text{ is a common independent set}] \leq (1 - 1/p)^k \leq e^{-k/p} .$$

The lemma now follows by taking the union bound over all possible pairs u, v ; the number of such pairs is upper bounded by $\binom{(p-1)m}{2}$. ◀

3.2 Hardness for online algorithms

Let $p = \lceil k/(27 \ln k) \rceil + 1$ and $m = k^3$. We will prove that the competitive ratio of any online algorithm is at least $k/(81 \ln k)$. We assume throughout that k is such that $k/(81 \ln k) > 1$. This is without loss of generality since the statement is trivial if $k/(81 \ln k) \leq 1$. We shall consider the following distribution of instances. Run Algorithm 1 with the parameters p and m to obtain sets S_1, \dots, S_{p-1} (and hidden elements $o^{(1)}, \dots, o^{(p-1)}$), and construct an input stream in which the elements of S_1 appear first (in any order) followed by those in S_2 and so on until the elements in S_{p-1} appear. We will show that any deterministic online algorithm ALG cannot be $((p-1)/3)$ -competitive on this distribution of instances. Theorem 6 then follows via Yao’s principle.

To analyze the competitive ratio of ALG, let O be the event that the output of ALG contains some element of $\{o^{(1)}, o^{(2)}, \dots, o^{(p-1)}\}$, and let S be the event that the instance is successful. Then, if we use $|ALG|$ to denote the size of the independent set outputted by ALG,

$$\mathbb{E}[|ALG|] \leq (1 - \Pr[\neg O, S]) \cdot k + 1 ,$$

where the inequality holds because any solution has size at most k , and if the algorithm fails to identify any element in $\{o^{(1)}, \dots, o^{(p-1)}\}$, then it can produce solution of size at most 1 for a successful instance.

Note that since $k/(81 \ln k) > 1$ we have that $p \leq 3 \cdot k/(27 \ln k) = k/(9 \ln k)$. Now, by Lemma 8 and the selection of p and m , we have

$$\Pr[S] \geq 1 - \binom{pm}{2} e^{-k/p} \geq 1 - k^8 \cdot e^{-9 \ln k} = 1 - 1/k .$$

To bound $\Pr[\neg O]$, note that the set S_r contains no information about $o^{(r)}$ because $o^{(r)}$ is selected uniformly at random from S_r . Moreover, when all m elements of S_r have been inspected in the stream, ALG keeps at most p independent elements. Any one of these elements is $o^{(r)}$ with probability at most p/m . In other words, the algorithm selects $o^{(r)}$ with probability at most p/m . Hence, by the union bound, the algorithm has selected any of the $p-1$ elements $\{o^{(1)}, \dots, o^{(p-1)}\}$ with probability at most $p/m \cdot (p-1) \leq 1/k$. We thus have $\Pr[\neg O] \geq 1 - 1/k$, and together with the above proved inequality $\Pr[S] \geq 1 - 1/k$, we get via the union bound

$$\mathbb{E}[|ALG|] \leq (1 - \Pr[\neg O, S]) \cdot k + 1 \leq (2/k) \cdot k + 1 = 3 .$$

Since the stream contains a solution $\{o^{(1)}, \dots, o^{(p-1)}\}$ of size $p-1$, this implies that ALG is not better than $((p-1)/3)$ -competitive, which in turn implies Theorem 6.

3.3 Hardness for streaming algorithms

Let ALG be a data stream algorithm for finding a set of maximum cardinality subject to k partition matroid constraints. Further suppose that ALG has the following properties:

- ALG uses memory M ;
- ALG outputs an α -approximate solution with probability at least $2/3$, where $\alpha \leq \frac{k}{32 \ln k}$ (note that α is also lower bounded by 1 since it is an approximation ratio).

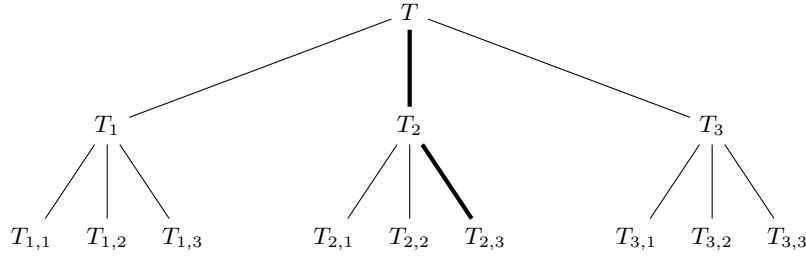
Select $p = \lceil 3 \cdot \alpha \rceil$, and let m be the smallest power of two such that $m \geq e^{k/(8\alpha)}$. Note that this selection satisfies

$$p \in [3 \cdot \alpha, 4 \cdot \alpha] , \quad m \in [e^{k/(8\alpha)}, 2e^{k/(8\alpha)}] , \quad \text{and} \quad 8 \cdot p \leq k \leq m .$$

We will use ALG to devise a protocol for the $\text{CHAIN}_p(m)$ problem that succeeds with probability at least $2/3$ and has communication complexity at most $M + p \log_2 m$. Combining this reduction with Theorem 5 then yields Theorem 2, i.e., that any such algorithm ALG must have a memory footprint M that is at least $\Omega(e^{k/(8\alpha)}/k^2)$.

3.3.1 Description of protocol

We use ALG to obtain Protocol 2 for $\text{CHAIN}_p(m)$. The protocol consists of two phases: a precomputation phase that is independent of the $\text{CHAIN}_p(m)$ instance, followed by a description of the messages of the players.



■ **Figure 1** A tree representation of the sets computed during precomputation for $m = p = 3$.

3.3.1.1 Precomputation phase

In the precomputation phase, the players use shared random coins⁶ to generate instances from the same distribution produced by Algorithm 1 for all possible values of $t^2, \dots, t^p \in [m]$ in an instance of $\text{CHAIN}_p(m)$. Specifically, first a set T is sampled from the same distribution as the set S_1 produced by Algorithm 1. The elements of T are then randomly permuted. For $j \in [m]$, we let $T(j)$ denote the j -th element in the obtained ordered set. The reason that the elements in T are randomly permuted is to make sure that for any fixed $t^2 \in [m]$, the element $T(t^2)$ is uniformly random, and thus, has the same distribution as $o^{(1)}$ in Algorithm 1. Following the choice of $S_1 = T$ and $o^{(1)} \in S_1$, Algorithm 1 proceeds to sample S_2 to be m random elements that are independent with respect to $o^{(1)}$. In the precomputation phase we do so for each possible element in T , i.e., we sample sets T_1, T_2, \dots, T_m , one for each possible choice of $t^2 \in [m]$. Then, for each such T_{t^2} we sample m sets $T_{t^2,1}, T_{t^2,2}, \dots, T_{t^2,m}$ for all possible choices of $t^3 \in [m]$ and so on. The sets constructed in the precomputation phase can thus naturally be represented by a tree, where each path from the root T to a leaf corresponds to a particular choice of $t^2, t^3, \dots, t^p \in [m]$. For $m = 3$ and $p = 3$, this tree is depicted in Figure 1. The thick path corresponds to the case of $t_2 = 2$ and $t_3 = 3$.

As described above, we randomly permute the sets so as to make sure that, for fixed $t^2, t^3, \dots, t^r \in [m]$, the distribution of $o^{(1)}$ is the same as that of $T(t^2)$ and, in general, the distribution of $o^{(i)}$ is the same as that of $T_{t^2, \dots, t^i}(t^{i+1})$. This gives us the following observation.

► **Observation 9.** Fix $t^2, t^3, \dots, t^r \in [m]$. Over the randomness of the precomputation phase, the elements $o^{(1)} = T(t^2), o^{(2)} = T_{t^2}(t^3), \dots, o^{(r-1)} = T_{t^2, \dots, t^{r-1}}(t^r)$ and the sets $S_1 = T, S_2 = T_{t^2}, \dots, S_{r-1} = T_{t^2, \dots, t^{r-1}}$ have the same distribution as the output of Algorithm 1.

The reason the players do this precomputation is that, after they have commonly agreed on the tree-structure of sets (which can be generated using the public coins), it requires little communication to decide on a “hard” instance generated from the same distribution as Algorithm 1. Indeed, Player r only needs to know t^2, \dots, t^r ($r \log_2 m$ bits of information) in-order to know the set T_{t^2, \dots, t^r} .

⁶ We note that the hardness result of $\text{CHAIN}_p(m)$ (Theorem 5) holds when the players have access to public coins, i.e., shared randomness. This is proved, e.g., in Theorem 3.3 of [9]. In general, Newman’s theorem [27] says that we can turn any public coin protocol into a private coin protocol with little (logarithmic) increase in communication.

■ **Protocol 2** Reduction from $\text{CHAIN}_p(m)$ to SMkM .

Precomputation

- 1: Let T be a uniformly random subset of $\mathcal{N} = [p]^k$ of size m .
- 2: Order the elements of T randomly, and let $T(j)$ denote the j -th element.
- 3: **for** $r = 2, \dots, p - 1$ and $t^2, \dots, t^r \in [m]$ **do**
- 4: Identify $o^{(1)} = T(t^2), o^{(2)} = T_{t^2}(t^3), \dots, o^{(r-1)} = T_{t^2, \dots, t^{r-1}}(t^r)$.
- 5: Let T_{t^2, \dots, t^r} be a uniformly random subset of $\{u \in \mathcal{N} \mid \forall 1 \leq i < r, 1 \leq j \leq k \ o_j^{(i)} \neq u_j\}$ of size m .
- 6: Order the elements of T_{t^2, \dots, t^r} randomly, and let $T_{t^2, \dots, t^r}(j)$ denote the j -th element.

Player P_r 's Algorithm for $r = 1, \dots, p - 1$

- 1: Initialize ALG with the received memory state (or initial state if first player).
- 2: Simulate ALG on the elements $T_r = \{T_{t^2, \dots, t^r}(j) \mid j \in [n] \text{ with } x_j^t = 1\}$ given in any order.
- 3: Send to P_{r+1} the values t^2, t^3, \dots, t^r and the memory state of ALG .

Player P_p 's Algorithm

- 1: If ALG with the received memory state returns an independent set of size at least 2, output “1-case”; otherwise, output “0-case”.
-

3.3.1.2 The messages of the players

After generating the (common) sets in the precomputation phase using the public coins, the players now proceed as follows. The first player receives as input x^1 and simulates ALG on the subset $\{T(j) \mid x_j^1 = 1\}$ of T corresponding to the 1-bits. These elements are given to ALG as a stream in any order. Player 1 then sends to Player 2 the message containing the state of ALG after processing this stream of elements.

The second player receives input x^2, t^2 and initializes ALG with the state received from the first player. Then, the elements $\{T_{t^2}(j) \mid x_j^2 = 1\}$ of T_{t^2} that correspond to 1-bits of x^2 are streamed to ALG in any order. Player 2 sends to Player 3 a message containing the state of ALG after processing these elements and the index t^2 . Player r , for $r = 3, \dots, p - 1$, proceeds similarly to Player 2: given input x^r, t^r , ALG is first initialized with the state received from the previous player, and then the elements $\{T_{t^2, \dots, t^r}(j) \mid x_j^r = 1\}$ are streamed to ALG in any order. Notice that Player r knows t^2, \dots, t^{r-1} from the message of the previous player, t^r, x^r from the input, and T_{t^2, \dots, t^r} from the precomputation phase, and so the set $\{T_{t^2, \dots, t^r}(j) \mid x_j^r = 1\}$ can be computed. Finally, Player r sends to Player $r + 1$ a message consisting of the indices t^2, \dots, t^r and the memory state of ALG .

The final player initializes ALG with the received state and asks ALG to return an independent set. If the independent set consists of at least two elements, Player p outputs “1-case”, and otherwise, the output is “0-case”.

3.3.2 Analysis

The messages sent by the players contain the memory state of ALG and at most $p - 2$ indices $t^2, \dots, t^{p-1} \in [m]$. The memory state of ALG is at most M bits by assumption, and each index requires $\log_2 m$ bits. The communication complexity of the protocol is, therefore, upper bounded by $M + p \log_2 m$.

To analyze the success probability of the protocol, we have the following lemma.

► **Lemma 10.** *The instance that the players stream to ALG satisfies the following:*

- *In the 1-case, the stream contains $p - 1$ independent elements.*
- *In the 0-case, with probability at least $2/3$, any two elements in the stream are dependent.*

Proof. In the 1-case, we have $x_{t^2}^1 = x_{t^3}^2 = \dots = x_{t^p}^{p-1} = 1$ and so the elements $T(t^2)$, $T_{t^2}(t^3)$, \dots , $T_{t^2, \dots, t^{p-1}}(t^p)$ are part of the stream. By definition, they form an independent set consisting of $p - 1$ elements.

In the 0-case, we have that $x_{t^2}^1 = x_{t^3}^2 = \dots = x_{t^p}^{p-1} = 0$ and so any two elements e, f in the stream belong to the set $S_1 \cup S_2 \dots, S_{p-1} \setminus \{o^{(1)}, o^{(2)}, \dots, o^{(p-1)}\}$, where $S_1 = T$, $o^{(1)} = T(t^2)$ and $S_j = T_{t^2, \dots, t^j}, o^{(j)} = T_{t^2, \dots, t^j}(t^{j+1})$ for $j = 2, \dots, p - 1$. By Observation 9, we can apply Lemma 8 to obtain that, with probability at least $1 - \binom{pm}{2} e^{-k/p}$, any two elements in the stream are dependent. The statement now follows since the selection of our parameters p, m and k implies

$$\binom{pm}{2} e^{-k/p} \leq m^4 e^{-k/p} \leq 16e^{k/(8\alpha)} \cdot e^{-k/(4\alpha)} = 16e^{-k/(8\alpha)} \leq 1/3 ,$$

where for the first inequality we used $p \leq m$, and for the second inequality we used that $p \leq 4 \cdot \alpha$ and $m \leq 2e^{k/(8\alpha)}$. The last inequality holds for $k \geq 3$ (the case of $k = 2$ can be ignored because it implies $k/(32 \ln k) < 1$, which makes Theorem 2 trivial). \blacktriangleleft

We now argue how the above lemma implies that Protocol 2 has a success probability of $2/3$ in both the 0-case and 1-case. For 0-case instances, we have with probability $2/3$ that any two elements are dependent. Hence, with that probability, there is no way for ALG to return an independent set with more than one element. Thus, the output of Player p is correct with probability at least $2/3$ in the 0-case. In the 1-case, the stream always contains a solution of value $p - 1$. By the assumption that ALG returns an α -approximate solution with probability at least $2/3$, ALG returns an independent set of size at least $(p - 1)/\alpha$ with probability at least $2/3$. This implies that Player p is correct in this case with probability $2/3$ since

$$(p - 1)/\alpha \geq (3 \cdot \alpha - 1)/\alpha \geq 2 .$$

Using ALG we have, thus, devised a protocol for $\text{CHAIN}_p(m)$ that is correct with probability $2/3$ and has a communication complexity that is upper bounded by $M + p \log_2 m$. By Theorem 5, we thus must have $(M + p \log_2 m) \geq m/(36p^2)$. Now using that $p \leq k/8$ and $e^{k/(8\alpha)} \leq m$, we get

$$M \geq \frac{m}{36p^2} - p \log_2 m \geq \frac{64}{36} \frac{m}{k^2} - k^2 \geq \frac{64}{36} \frac{e^{k/(8\alpha)}}{k^2} - k^2 ,$$

which is $\Omega(e^{k/(8\alpha)}/k^2)$ since by assumption on α we have $e^{k/(8\alpha)} \geq k^4$. We have thus proved that the memory usage M of ALG must be at least $\Omega(e^{k/(8\alpha)}/k^2)$ as required by Theorem 2.

References

- 1 Naor Alaluf, Alina Ene, Moran Feldman, Huy L. Nguyen, and Andrew Suh. Optimal streaming algorithms for submodular maximization with cardinality constraints. In *ICALP*, pages 6:1–6:19, 2020. doi:10.4230/LIPIcs.ICALP.2020.6.
- 2 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 671–680, 2014.
- 3 Gruiă Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

- 4 A. Chakrabarti. Lower bounds for multi-player pointer jumping. *Electronic Colloquium on Computational Complexity*, 14, 2007.
- 5 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154(1):225–247, December 2015.
- 6 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming*, pages 318–330, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- 7 G. Cormode, J. Dark, and C. Konrad. Independent sets in vertex-arrival streams. In *Proceedings of 46th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 45:1–45:14, 2019.
- 8 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.
- 9 M. Feldman, A. Norouzi-Fard, O. Svensson, and R. Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory on Computing (STOC)*, pages 1363–1374, 2020.
- 10 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 730–740, 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/d1f255a373a3cef72e03aa9d980c7eca-Abstract.html>.
- 11 Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions–II. *Mathematical Programming*, 8:73–87, 1978.
- 12 Ran Haba, Ehsan Kazemi, Moran Feldman, and Amin Karbasi. Streaming submodular maximization under a k -set system constraint. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 3939–3949, 2020. URL: <http://proceedings.mlr.press/v119/haba20a.html>.
- 13 E. Hazan, S. Safra, and O. Schwarz. On the complexity of approximating k -set packing. *Computational Complexity*, 15(1):20–39, 2006. 2006.
- 14 Chien-Chung Huang, Naonori Kakimura, Simon Mauras, and Yuichi Yoshida. Approximability of monotone submodular function maximization under cardinality and matroid constraints in the streaming model. *CoRR*, abs/2002.05477, 2020. [arXiv:2002.05477](https://arxiv.org/abs/2002.05477).
- 15 Chien-Chung Huang, Theophile Thiery, and Justin Ward. Improved multi-pass streaming algorithms for submodular maximization with matroid constraints. *CoRR*, abs/2102.09679, 2021. [arXiv:2102.09679](https://arxiv.org/abs/2102.09679).
- 16 Chien-Chung Huang and Justin Ward. Fpt-algorithms for the l -matchoid problem with linear and submodular objectives. *CoRR*, abs/2011.06268, 2020. [arXiv:2011.06268](https://arxiv.org/abs/2011.06268).
- 17 Tom A. Jenkyns. The efficacy of the “greedy” algorithm. In *South Eastern Conference on Combinatorics, Graph Theory and Computing*, pages pages 341–350, 1976.
- 18 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1874–1893. SIAM, 2021. doi:10.1137/1.9781611976465.112.
- 19 Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 3311–3320, 2019. URL: <http://proceedings.mlr.press/v97/kazemi19a.html>.
- 20 Bernhard Korte and Dirk Hausmann. An analysis of the greedy heuristic for independence systems. *Annals of Discrete Math.*, 2:65–74, 1978.
- 21 Andreas Krause. Submodularity in machine learning. <http://submodularity.org/>.

- 22 Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Math. Oper. Res.*, 35(4):795–806, 2010. doi: 10.1287/moor.1100.0463.
- 23 Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1914–1933, 2021. doi: 10.1137/1.9781611976465.114.
- 24 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory Comput. Syst.*, 63(7):1595–1619, 2019.
- 25 George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.
- 26 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.
- 27 Ilan Newman. Private vs. common random bits in communication complexity. *Inf. Process. Lett.*, 39(2):67–71, 1991.
- 28 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 1 edition, 2003.

There and Back Again: On Applying Data Reduction Rules by Undoing Others

Aleksander Figiel ✉

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Vincent Froese ✉

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

André Nichterlein ✉ 

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Rolf Niedermeier 

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Abstract

Data reduction rules are an established method in the algorithmic toolbox for tackling computationally challenging problems. A data reduction rule is a polynomial-time algorithm that, given a problem instance as input, outputs an equivalent, typically smaller instance of the same problem. The application of data reduction rules during the preprocessing of problem instances allows in many cases to considerably shrink their size, or even solve them directly. Commonly, these data reduction rules are applied exhaustively and in some fixed order to obtain irreducible instances. It was often observed that by changing the order of the rules, different irreducible instances can be obtained. We propose to “undo” data reduction rules on irreducible instances, by which they become larger, and then subsequently apply data reduction rules again to shrink them. We show that this somewhat counter-intuitive approach can lead to significantly smaller irreducible instances. The process of undoing data reduction rules is not limited to “rolling back” data reduction rules applied to the instance during preprocessing. Instead, we formulate so-called backward rules, which essentially undo a data reduction rule, but without using any information about which data reduction rules were applied to it previously. In particular, based on the example of VERTEX COVER we propose two methods applying backward rules to shrink the instances further. In our experiments we show that this way smaller irreducible instances consisting of real-world graphs from the SNAP and DIMACS datasets can be computed.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Branch-and-bound

Keywords and phrases Kernelization, Preprocessing, Vertex Cover

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.53

Related Version A full version of the paper is available at: <https://arxiv.org/abs/2206.14698> [10]

Supplementary Material *Software (Implementation)*: <https://git.tu-berlin.de/afigiel/undo-vc-drr>, archived at `swb:1:dir:54f57455bc1dc965f9315a1cc800cfd768bbdac3`

Funding Aleksander Figiel: Supported by DFG project “MaMu” (NI369/19).

Acknowledgements This work is based on the first author’s master’s thesis.

In memory of Rolf Niedermeier, our colleague, friend, and mentor, who sadly passed away before this paper was published.



© Aleksander Figiel, Vincent Froese, André Nichterlein, and Rolf Niedermeier; licensed under Creative Commons License CC-BY 4.0

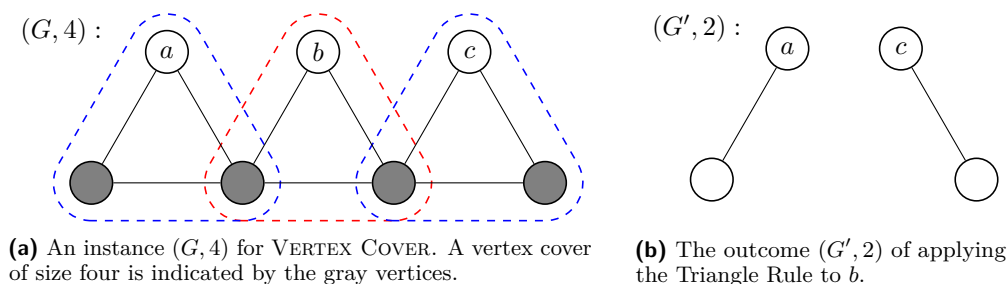
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, 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



■ **Figure 1** An example of how the order of reduction rules can affect the final instance. A VERTEX COVER instance with $k = 4$ is depicted left. By applying the Triangle Rule to b (red triangle), we obtain a graph with two edges (right). However, applying the rule to a and then c (therefore removing the two blue triangles) yields the smaller graph consisting only of the vertex b .

1 Introduction

Kernelization by means of applying data reduction rules is a powerful (and often essential) tool for tackling computationally difficult (e. g. NP-hard) problems in theory and in practice [11, 2]. A data reduction rule is a polynomial-time algorithm that, given a problem instance as input, outputs an “equivalent” and often smaller instance of the same problem. One may think of data reduction as identifying and removing “easy” parts of the problem, leaving behind a smaller instance containing only the more difficult parts. This instance can be significantly smaller than the original instance [1, 3, 16, 13, 18] which makes other methods like branch&bound algorithms a viable option for solving it. In this work, we apply existing data reduction rules “backwards”, that is, instead of smaller instances we produce (slightly) larger instances. The hope herein is, that this alteration of the instance allows subsequently applied data reduction rules to further shrink the instance, thus, producing even smaller instances than with “standard” application of data reduction rules.

We consider the NP-hard VERTEX COVER – the primary “lab animal” in parameterized complexity theory [9] – to illustrate our approach and to exemplify its strengths.

VERTEX COVER [12]

Input: An undirected graph $G = (V, E)$ and $k \in \mathbb{N}$.

Question: Is there a set $S \subseteq V$, $|S| \leq k$, covering all edges, i. e., $\forall e \in E: e \cap S \neq \emptyset$?

VERTEX COVER is a classic problem of computational complexity theory and one of Karp’s 21 NP-complete problems [15]. We remark that for presentation purposes we use the decision version of VERTEX COVER. All our results transfer to the optimization version (which our implementation is build for).

To explain our approach, assume that all we have is the following data reduction rule:

► **Reduction Rule 1** (Triangle Rule [9]). *Let $(G = (V, E), k)$ be an instance of VERTEX COVER and $v \in V$ a vertex with exactly two neighbors u and w . If the edge $\{u, w\}$ exists, then delete v , u , and w from the graph (and their incident edges), and decrease k by two.*

As illustrated in Figure 1, there are two options to apply Rule 1 for the instance $(G, 4)$. Picking the “bad” option, that is, applying it to b yields the instance $(G', 2)$. Note that (the correctness of) Rule 1 implies that $(G, 4)$ and $(G', 2)$ are “equivalent”, that is, either both of them are yes-instances or none of them are. Hence, if we have the instance $(G', 2)$ on the right side (either through the “bad” application of Rule 1 or directly as input), then we can

apply Rule 1 “backwards” and obtain the equivalent but larger instance $(G, 4)$ on the left side. Then, by applying Rule 1 to a and c , we can arrive at the edge-less graph $(\{b\}, \emptyset)$, thus “solving” the triangle-free instance $(G', 2)$ by only using the Triangle Rule.

More formally, the setting can be described as follows: A data reduction rule for a problem L is a polynomial-time algorithm which *reduces* an instance x to an equivalent instance x' , that is, $x \in L$ if and only if $x' \in L$. A set of data reduction rules thus implicitly partitions the space of all instances into classes of equivalent instances (two instances are in the same equivalence class if one of them can be obtained from the other by applying a subset of the data reduction rules). The more data reduction rules we have, the fewer and larger equivalence classes we have. Now, the overall goal of data reduction is to find the *smallest* instance in the same equivalence class. We demonstrate two approaches tailored towards (but not limited to) graph problems to tackle this task.

Let us remark that while there are some analogies to the branch&bound paradigm (searching for a solution in a huge search space), there are also notable differences: A branching rule creates several instances of which at least one is guaranteed to be equivalent to the original one. The problem is that, a priori, it is not known which of these instances is the equivalent one. Hence, one has to “solve” all instances before learning the solution. In contrast, our setting allows stopping at *any* time as the currently handled instance is *guaranteed* to be equivalent to the starting instance. This allows for considerable flexibility with respect to possible combinations with other approaches like heuristics, approximation or exact algorithms.

Related Work. Fellows et al. [9] are closest to our work. They propose a method for automated discovery of data reduction rules looking at rules that replace a small subgraph by another one. They noticed that if a so called *profile* (which is a vector of integers) of the replaced subgraph and of the one taking its place only differ by a constant in each entry, then this replacement is a data reduction rule. To then find data reduction rules, one can enumerate all graphs up to a certain size and compute their profile vectors. The downside of this approach is that in order to apply the automatically found rules one has solve a (computationally challenging) subgraph isomorphism problem or manually design new algorithms for each new rule.

VERTEX COVER is extensively studied from the the viewpoint of data reduction and kernelization; see Fellows et al. [9] for an overview. Akiba and Iwata [3] and Hesse et al. [14] provide exact solvers that include an extensive list of data reduction rules. The solver of Hesse et al. [14] won the exact track for VERTEX COVER at the 4th PACE implementation challenge [7]. We provide a list of data reduction rules for VERTEX COVER in the full version [10].

Alexe et al. [4] experimentally investigated by how much the so-called Struction data reduction rule for INDEPENDENT SET can shrink small random graphs. The Struction data reduction rule can always be applied to any graph and decreases the stability number¹ of a graph by one, but may increase the number of vertices quadratically each time it is applied. Gellner et al. [13] proposed a modification of the Struction rule for the MAXIMUM WEIGHTED INDEPENDENT SET problem. They first restricted themselves to only applying data reduction rules if they do not increase the number vertices in the graph, which they call the reduction phase. They then compared this method to an approach which allows

¹ An independent set is a set of pairwise nonadjacent vertices. The stability number or the independence number of a graph G is the size of a maximum independent set of G .

their modified Struction rule to also increase the number of vertices in the graph by a small fraction, which they call the blow-up phase. The experiments showed, that repetitions of the reduction and blow-up phase can significantly shrink the number of vertices compared to just the reduction phase.

Ehrig et al. [8] defined the notion of confluence from rewriting systems theory for kernelization algorithms. Intuitively, confluence in kernelization means that the result of applying a set of data reduction rules exhaustively to the input always results in the same instance, up to isomorphism, regardless of the order in which the rules were applied. It turns out that for our approach to work we require non-confluent data reduction rules.

Our Results. In Section 3, we provide two concrete methods to apply existing data reductions rules “backwards” and “forwards” in order to shrink the input as much as possible. We implemented these methods and applied them on a wide range of data reduction rules for VERTEX COVER. Our experimental evaluations are provided in Section 4 where we use our implementation on instances where the known data reductions rules are not applicable. Our implementation can also be used to preprocess a given graph G and it returns the smallest found kernel K after a user specified amount of time. Moreover, the implementation can translate a provided solution S_K for the kernel into a solution S_G for the initial instance such that $|S_G| \leq |S_K| + d$ where $d := \tau(G) - \tau(K)$ is the difference between the sizes of minimum vertex covers of G and K . Thus, if a minimum vertex cover for K is provided it will be translated into a minimum vertex cover for G .

2 Preliminaries

We use standard notation from graph theory and data reduction. In this work, we only consider simple undirected graphs G with vertex set $V(G)$ and edge set $E(G) \subseteq \{\{v, w\} \mid v, w \in V(G), v \neq w\}$. We denote by n and m the number of vertices and edges, respectively. For a vertex $v \in V(G)$ the open (closed) neighborhood is denoted with $N_G[v]$ ($N_G(v)$). For a vertex subset $S \subseteq V(G)$ we set $N_G[S] := \bigcup_{v \in S} N_G[v]$. When in context it is clear which graph is being referred to, the subscript G will be omitted in the subscripts.

Data Reduction Rules. We use notions from kernelization in parameterized algorithmics [11]. However, we simplify the notation to unparameterized problems. A data reduction rule for a problem $L \subseteq \Sigma^*$ is a polynomial-time algorithm, which *reduces* an instance x to an equivalent instance x' . We call an instance x *irreducible* with respect to a data reduction rule, if the data reduction rule does not change the instance x any further (that is, $x' = x$). The property that the data reduction rule returns an equivalent instance is called *safeness*. We call an instance obtained from applying data reduction rules *kernel*.

Often, data reduction rules can be considered nondeterministic, because a data reduction rule could change the input instance in a variety of ways (e. g., see the example in Section 1). To highlight this effect and to avoid confusion, we introduce the term *forward rule*. A forward rule is a subset $R_{\mathcal{A}} \subseteq \Sigma^* \times \Sigma^*$ associated with a nondeterministic polynomial-time algorithm \mathcal{A} , where $(x, y) \in R_{\mathcal{A}}$ if and only if y is one of the possible outputs of \mathcal{A} on input x . Intuitively, a forward rule captures all possible instances that can be derived from the input instance by applying a data reduction rule a single time. To define what it means to “undo” a reduction rule, we introduce the term *backward rule*. A backward rule is simply the converse relation $R_{\mathcal{A}}^{-1} := \{(y, x) \mid (x, y) \in R_{\mathcal{A}}\}$ of some forward rule $R_{\mathcal{A}}$.

Confluence. A set of data reduction rules is said to be terminating, if for all instances \mathcal{I} , the data reduction rules in the set cannot be applied to the instance \mathcal{I} infinitely many times. A set of terminating data reduction rules is said to be confluent if any exhaustive way of applying the rules yields a unique irreducible instance, up to isomorphism [8]. It is not hard to see that given a set of confluent data reduction rules, undoing any of them is of no use, because subsequently applying the data reduction rules will always result in the same instance.

► **Lemma 2.1.** *Let \mathcal{R} be a confluent set of forward rules, \mathcal{I} a problem instance, and $\mathcal{I}^{\mathcal{R}}$ the unique instance obtained by applying the rules in \mathcal{R} to \mathcal{I} .*

Further let $\overline{\mathcal{R}} = \{R^{-1} \mid R \in \mathcal{R}\}$ be the set of backward rules corresponding to \mathcal{R} . Let \mathcal{I}^- be an instance that was derived by applying some rules from $\mathcal{R} \cup \overline{\mathcal{R}}$. Then applying the rules in \mathcal{R} exhaustively in any order to \mathcal{I}^- will yield an instance isomorphic to $\mathcal{I}^{\mathcal{R}}$.

Proof. We prove the lemma by induction over the number of times backward rules were applied to obtain \mathcal{I}^- .

Base case: if no backward rules were applied to obtain \mathcal{I}^- , then by exhaustively applying \mathcal{R} we will obtain an instance isomorphic to $\mathcal{I}^{\mathcal{R}}$.

Inductive step: Assume only n backward rules were applied to obtain \mathcal{I}^- . Consider the sequence $\mathcal{I} = \mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_{\ell-1}, \mathcal{I}_{\ell} = \mathcal{I}^-$ of instances which were on the way from \mathcal{I} to \mathcal{I}^- during the application of the rules in $\mathcal{R} \cup \overline{\mathcal{R}}$. Further let \mathcal{I}_i be the instance in the sequence after applying the n 'th backward rule R^{-1} . After applying R to \mathcal{I}_i the instance \mathcal{I}_{i-1} can be obtained, which was obtained by using only $n - 1$ backward rules. Because after R^{-1} only rules in \mathcal{R} were applied, which are confluent, we may instead assume that the first rule which was applied after R^{-1} was R . Consequently, by the induction hypothesis an instance isomorphic to $\mathcal{I}^{\mathcal{R}}$ will be obtained by exhaustively applying the rules in \mathcal{R} . ◀

3 Two Methods for Achieving Smaller Kernels

We apply data reduction rules “back and forth” to obtain an equivalent instance as small as possible. This gives rise to a huge search space for which exhaustive search is prohibitively expensive. Thus, some more sophisticated search procedures are needed. In this section, we propose two approaches which we call the Find and the Inflate-Deflate method. We implemented and tested both approaches; the experimental results are presented in Section 4.

The Find method (Section 3.1) shrinks the naive search tree with heuristic pruning rules in order to identify sequences of forward and backward rules, which when applied to the input instance, produce a smaller equivalent instance. We employ this method primarily to find such sequences which are *short*, so that those sequences may actually be used to formulate *new* data reduction rules. It naturally has a *local* flavor in the sense that changes of one iteration are bound to a (small) part of the input graph.

The Inflate-Deflate method (Section 3.2) is much less structured. It randomly applies backward rules until the instance size increased by a fixed percentage. Afterwards, all forward rules are applied exhaustively. If the resulting instance is smaller, then the process is repeated; otherwise, all changes are reverted.

3.1 Find Method

For finding sequences of forward and backward rules which when applied to the input instance produce a smaller instance, we propose a structured search approach based on recursion. Let \mathcal{I} be the input instance and let $\mathcal{F}_{\mathcal{I}}$ be the set of all instances reachable via one forward

or backward rule, i. e., $\mathcal{F}_{\mathcal{I}} = \{\mathcal{I}' \mid (\mathcal{I}, \mathcal{I}') \in R \text{ for any forward or backward rule } R\}$. If the input instance is irreducible with respect to the set of forward rules, then only backward rules will be applicable. We branch into $|\mathcal{F}_{\mathcal{I}}|$ cases where, for each $\mathcal{I}' \in \mathcal{F}_{\mathcal{I}}$, we try to recursively find forward and backward rules applicable to \mathcal{I}' and branch on each of them. This is repeated until some maximum recursion depth is reached or the sequence of forward and backward rule applications results in a smaller instance. In the latter case, the sequence of applied rules can be thought of as a new data reduction rule.

Note that the search space is immense: For example, consider the backward rule corresponding to Rule 1, which inserts three vertices u , v , and w , makes them pairwise adjacent, and inserts an arbitrary set of edges between u and v and the original vertices. Thus, for each original vertex, there are four options (make it adjacent to u , to v , to u and v , or neither). This results in 4^n options for applying just this one single backward rule. Hence, it is clear that we have to introduce suitable methods to cut off large parts of the resulting search tree. To this end, we heavily rely on the observation that many reduction rules have a “local flavor”.

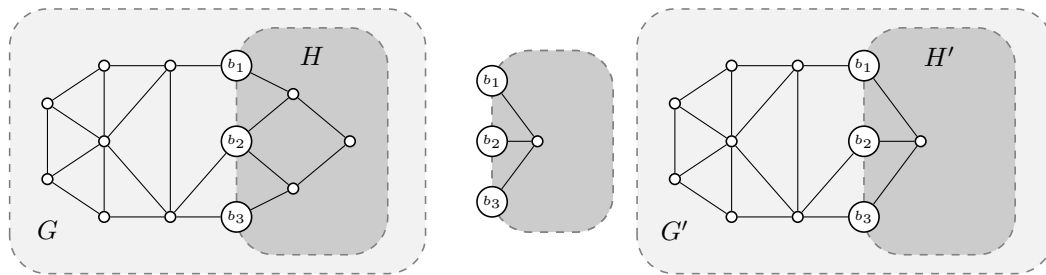
Region of Interest. To avoid a very large search space we only consider applying forward and backward rules “locally”. For this, we define a “region of interest”, which for graph problems is a set $X \subseteq V$. Any forward or backward rule must only be applied within the region of interest. We initially start with a very modest region of interest, namely $X = \{v\}$ for all $v \in V$. Thus, we work with n regions of interest per graph, each one considered separately.

Forward and backward rules are allowed to leave the region of interest only if at least one vertex that is “relevant” for the rule is within the region of interest. Moreover, the region of interest is allowed to “grow” as rules are applied. This is because applications of rules might cause further rules to become applicable. For example, rules might become applicable to the neighbors of vertices modified by the previously applied rules.

Specifically, let M be the set of “modified” vertices, which are new vertices or vertices which gained or lost an edge as a result of applying a forward or backward rule, and let D be the set of vertices it deleted. In the case of VERTEX COVER, we suggest to expand the region of interest to $(X \cup N[M]) \setminus D$ after each rule application. The majority of forward rules for VERTEX COVER are “neighborhood based”. Of course, the region of interest could be expanded even further, e. g., by extending it by $N^2[M]$ instead, but this will of course increase the search space. With a larger region of interest we might find more reduction rules, but at the cost of higher running time. Additionally, the found reductions may be more complex, and modify a large subgraph and are therefore difficult to analyze or implement.

We will call the above method, which recursively applies forward and backward rules one by one restricted to only the region of interest, the Find method. Note that we “accept” a sequence of forward and backward rules if it decreases the number of vertices or the parameter k , without increasing either. We have done so to find only “nice” data reduction rules where there is no trade-off between decreasing the parameter k or the number of vertices. However, different conditions to “accept” a sequence of forward and backward rules are also possible. For example, by only requiring that the number of vertices or edges is decreased.

Graph modification. Our implementation of the Find method outputs sequences of rules which are able to shrink the graph. However, just knowing which rules and in what order they were applied may not be very helpful in understanding the changes made by the rules. Specifically, it does not show how the rules were applied.



■ **Figure 2** A graph modification example corresponding to the application of the Degree-2 Folding Rule that applies to vertices with exactly two non-adjacent neighbors (the rule merges the degree-two vertex and its neighbors). The set $B = \{b_1, b_2, b_3\}$ is the boundary of the graph modification. Notice that only vertices in B (the boundary) are adjacent to both vertices in $V(H)$ and $V(G) \setminus V(H)$.

For this reason, we introduce the notion of a *graph modification* based on the ideas by Fellows et al. [9]. A graph modification encodes how a single or multiple data reduction rules have changed a graph. We say the *boundary* of a subgraph H of a graph G is the set B of all vertices in $V(H)$ whose neighborhood in G contains vertices in $V(G) \setminus V(H)$.

► **Definition 3.1.** A graph modification is a 4-tuple of graphs (G, H, H', G') with the following properties

- $B = V(H') \cap V(G)$,
- H is a subgraph of G with boundary B , and
- G' is derived from G by deleting all vertices in $V(H) \setminus B$ and all edges among B from G and then adding the vertices in $V(H') \setminus B$ and adding all edges from H' .

See Figure 2 for an example of a graph modification. The Find method can be extended such that in addition to printing rule sequences it also outputs the graph modifications corresponding to each application of a rule from the found sequences. This can be achieved by keeping track of newly created or deleted vertices and edges by the applied rules.

Isomorphism. It may happen that two or more rule sequences change an instance in the same way, that is, they produce isomorphic graphs from the same input instance. In order to avoid double counting, we implemented an isomorphism test to avoid these issues; see the full version [10] for details.

The Find and Reduce Method. The Find and Reduce Method is just a small variation of Find. Instead of only searching for sequences of forward and backward rules which shrink the instance, upon finding such a sequence it is also immediately applied to the instance. The search for more sequences continues with the smaller instance. This method serves the dual purpose of both finding sequences of rules which shrink the instance (and therefore also finding reduction rules), but also that of producing a smaller irreducible instance. Another potential advantage of this method is that it finds only the rules which were used to produce the smaller instance, and may therefore be more practical than the ones found by Find. Recall, that Find only searches for reduction rules applicable directly to the input instance. Perhaps by applying a single such sequence, different sequences are needed to shrink the remaining instance further.

3.2 Inflate-Deflate

Both the Find and the Find and Reduce method only change a small part of the instance. It may, however, be necessary to change large parts of an instance before it can be shrunk to a size smaller than it was originally. For this reason, we propose the Inflate-Deflate inspired by the Cyclic Blow-Up Algorithm by Gellner et al. [13].

Essentially, Inflate-Deflate iteratively runs two phases: First, in the inflation phase, randomly applies a set of backward rules to the instance until it becomes some fixed percentage α larger than it was initially. Then, in the deflation phase, exhaustively apply a set of forward rules and repeat with the inflation phase again. Our implementation has the termination condition $|V| = 0$ which may never be met. Thus a timeout or limit on the number of iterations has to be specified. The inflation factor α can be freely set to any value greater zero. We investigate the effect of different inflation factors in Section 4 for values of α between 10% and 50%.

In our deflate procedure, we apply the set of forward rules exhaustively in a particular way: An applicable forward rule is randomly chosen, and then it is randomly applied to the instance, but only once. Afterwards another applicable rule is chosen randomly, and this is repeated until the instance becomes irreducible with respect to all forward rules. This randomized exhaustive application of the forward rules ensures that the rules are not applied in a predefined way, and different “interactions” of the rules are tested. For example, consider the case where in the inflate phase the Backward Degree-2 Folding Rule was applied, then likely one would not want to immediately exhaustively apply the (Forward) Degree-2 Folding Rule (see Figure 2) which could in effect directly cancel the changes made by that backward rule. Furthermore, in this way, each of the forward rules has a chance to be applied. This avoids any potential problems due to an inconvenient fixed rule order.

Because large sections of an instance are modified at once, no short sequences of forward and backward rules can be extracted from this method. As a result, it is unlikely that new reduction rules could be learned this way. However, the method produces smaller irreducible instances as we will see in Section 4.

Local Inflate-Deflate. Within Inflate-Deflate it may happen that if we inflate the instance and then deflate it again, often the resulting instance is larger. In such cases the number of “negative” changes to the instance outweigh the number of “positive” changes. To increase the success probability one may try to lower the inflation factor, however then it can also happen that positive changes are less likely.

An alternate way to try to increase the success probability, is to apply backward rules within a randomly chosen subgraph rather than the whole graph. For example, this subgraph could be the set of all vertices with some maximum distance to a randomly chosen vertex. Backward rules are then applied until the subgraph becomes larger by a factor of α , instead of the whole graph.

4 Experimental Evaluation

In this section, we describe the experiments which we performed based on an implementation of the methods described in Section 3.

■ **Table 1** A glossary of the forward and backward rules which were used in our implementation. Note that we apply the Structon Rule only if it does not increase k . In the columns named alias we provide shortened names for the rules. We refer to the full version of the paper [10] for detailed descriptions of these rules.

Forward rules		Backward rules	
Alias	Full name	Alias	Full name
Deg0	Degree-0	Undeg2	Backward Degree-2 Folding
Deg1	Degree-1	Undeg3	Backward Degree-3 Independent Set
Deg2	Degree-2 Folding	Uncn	Backward 2-Clique Neighborhood (special case)
Deg3	Degree-3 Independent Set	Undom	Backward Domination
Dom	Domination	Ununconf	Backward Unconfined
Unconf	Unconfined- κ ($\kappa = 4$)	OE_Ins	Optional Edge Insertion
Desk	Desk		
CN	2-Clique Neighborhood		
OE_Del	Optional Edge Deletion		
Struct	Structon ($k' \leq k$)		
Magnet	Magnet		
LP	LP		

4.1 Setup

Computing Environment. All our experiments were run on a machine running Ubuntu 18.04 LTS with the Linux 4.15 kernel. The machine is equipped with an Intel® Xeon® W-2125 CPU, with 4 cores and 8 threads² clocked at 4.0 GHz and 256GB of RAM.

Datasets. For our experiments we used three different datasets: DIMACS, SNAP and PACE; the lists of graphs are given in the full version [10]. The DIMACS and SNAP datasets are commonly used for graph-based problems, including VERTEX COVER [3, 16, 13]. We have used the instances from the 10th DIMACS Challenge [5], specifically from the Clustering, Kronecker, Co-author and Citation, Street Networks, and Walshaw subdatasets. In total these are 82 DIMACS instances. From the SNAP Dataset Collection [17] we have used the graphs from from the Social, Ground-Truth Communities, Communication, Collaboration, Web, Product Co-purchasing, Peer-to-peer, Road, Autonomous systems, Signed and Location subdatasets. In total we obtained 52 SNAP instances. Additionally, we used a dataset which was used specifically for benchmarking VERTEX COVER solvers in the 2019 PACE Challenge [7]. We used the set of 100 private instances [6], which were used for scoring submitted solvers.

Preprocessing and Filtering. We apply some preprocessing to our datasets. We obtain simple, undirected graphs by ignoring any potential edge direction or weight information from the instances and by deleting self-loops. To these graphs we apply the forward rules, see Table 1 and the full version [10] for an overview: Deg1, Deg2, Deg3, Unconf, Cn, LP, Struct, Magnet and Oe_delete exhaustively in the given order. We note that the kernels obtained this way always had fewer vertices than the kernels obtained with the data reduction suite used by Akiba and Iwata [3] and also Hesse et al. [14].

² All our implementations are single-threaded.

We filter out graphs that became empty as a result of applying these rules. These were 41 DIMACS, 33 SNAP and 12 PACE instances. Furthermore, we discard graphs which after applying these rules still had more than 50,000 vertices. These were 12 DIMACS and 3 SNAP instances. Because the PACE instances may also contain instances from the other two datasets, we have tested the graphs for isomorphism. We have found one PACE instance to be isomorphic to a SNAP instance (p2p-Gnutella09), which was already excluded, because it was shrunk to an empty graph.

In total, we are left with 31 DIMACS, 16 SNAP and 88 PACE graphs with at most 50,000 vertices – all of these graphs are irreducible with respect to the forward rules. When referring to the datasets DIMACS, SNAP and PACE we will be referring to these kernelized and filtered instances.

Implementation. The major parts of our implementation are written in C++11 and compiled using version 7.5 of g++ using the -O2 optimization flag. Smaller parts, such as scripts for visualization or automation were written in Python 3.6 or Bash. We provide the source code for our implementation at <https://git.tu-berlin.de/afigiel/undo-vc-drr>. This implementation contains our Find and Inflate-Deflate method, together with the two small variations Find and Reduce, and local Inflate-Deflate. Find uses the local isomorphism test which we describe in Section 3.1, and output a description of the graph modification in addition to the found sequences.

We also provide a VERTEX COVER solver implementation with all our forward rules implemented, and some new data reduction rules which are explained later in this section. The solver is based on the branch-and-reduce paradigm and is very similar to the VERTEX COVER solver by Akiba and Iwata [3]. Moreover, we provide a lifting algorithm that can transform solutions for the kernelized instances into solution for the original instances. We also provide a Python script that is used to visualize the graph modifications of the sequences of forward and backward rules that are output by our methods. However, our focus in this section is on the Find and Inflate-Deflate method.

Methodology. We have implemented our Find and Inflate-Deflate methods together with their two variations: Find and Reduce and local Inflate-Deflate. Almost all forward and backward rules described in the full version [10] are used by these methods, see Table 1 for an overview.

All rules increase or decrease k , but do not need to know k in advance. This allows us to run Find and Inflate-Deflate on all graphs, without having to specify a value for k . Instead, we set $k = 0$ for all instances. In the final instance (G', k') computed from $(G, k = 0)$ we will have $\tau(G') - k' = \tau(G)$, where $\tau(G)$ denotes the vertex cover number of G . For graphs G' which become empty, $-k'$ is the vertex cover number of the original graph G .

We set a maximum recursion limit for Find such that only sequences of at most two or three rules are found. We will refer to FAR2 and FAR3 as the Find and Reduce method which only searches for sequences of at most two or three forward and backward rules, respectively.

We used inflation ratios α equal to 10, 20 and 50 percent, and we will refer to the different Inflate-Deflate configurations as ID10, ID20, and ID50, respectively. We also tested our local Inflate-Deflate method with $\alpha = 20\%$, which we have found to work best in preliminary experiments, which we will refer to as LID20.

All these configurations were tested on the three datasets with a maximum running time of one hour.

	Deg1	Deg2	Deg3	Desk	Dom	Unconf	CN	LP	Struct	Magnet	OE-Del
Deg1	Gray										
Deg2		Gray									
Deg3			Gray								
Desk				Gray							
Dom					Gray						
Unconf						Gray					
CN							Gray				
LP								Gray			
Struct									Gray		
Magnet										Gray	
OE-Del											Gray

■ **Figure 3** A matrix depicting which pairs of forward rules we have found to be non-confluent. A cell corresponding to a row rule R_a and a column rule R_b is colored white if the set $\mathcal{R} = \{R_a, R_b, \text{Deg0}\}$ is not confluent. The Degree-0 rule is always included to not take into account differences in number of isolated vertices left after applying the rules. Gray cells correspond to sets which may be confluent, meaning that no counter-example was found for them.

4.2 Results

Confluence. Confluence can be proved using for example conflict pair analysis [8], which is not straightforward and does not work for all types of data reduction rules. However, disproving confluence is potentially much easier, as it suffices to find one example where applying the rules in different order yields different instances.

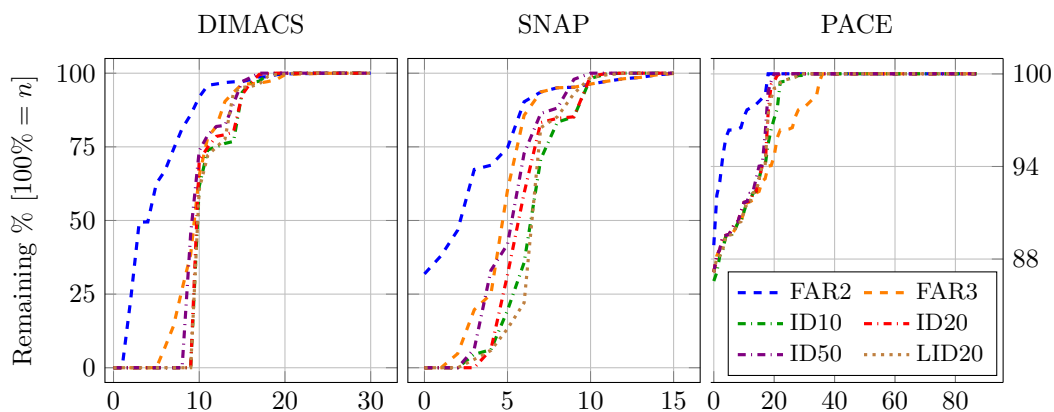
For each pair of forward rules in Table 1, we tested on the set of all graphs with at most 9 vertices³ whether we can obtain irreducible, but non-isomorphic graphs by randomly applying the two rules to the same graph. More precisely, we test whether a set $\mathcal{R} = \{R_a, R_b, \text{Deg0}\}$ is confluent for two forward rules R_a and R_b . We include the Degree-0 Rule in these sets, because a difference in the number of isolated vertices is only a minor detail which we do not wish to take into account. We note that, the inclusion of the Degree-0 Rule in these sets was never the reason that a set was not confluent.

Our results are summarized in Figure 3. The figure clearly shows that most pairs of forward rules are not confluent. This means that the relative order of these rules may affect the final instance.

Instance Shrinking. Next, we demonstrate how much the Find and Reduce and Inflate-Deflate methods were able to shrink the irreducible DIMACS, SNAP, and PACE graphs, which were obtained by exhaustively applying a set of forward rules. The results are summarized in Figure 4 and Table 2.

Notably, ten DIMACS and four SNAP instances were reduced to an empty graph by the Inflate-Deflate methods. Five further DIMACS graphs shrank to around 80% of their size, and half of the DIMACS graphs did not really shrink at all. We conclude that the Inflate-Deflate approach seems to either work really well or nearly not at all for a given instance.

³ We obtained these graphs from Brendan McKay's website <https://users.cecs.anu.edu.au/~bdm/data/graphs.html>



■ **Figure 4** Cactus plots depicting how many irreducible instances were shrunk to a given fraction of their size (measured in vertices). The order of the instances is chosen for each configuration such that the size fractions of the instances are increasing. Note that the y-axis for DIMACS and SNAP start at zero (i. e. instances reduced to the empty graph); the y-axis for PACE does *not*.

■ **Table 2** Summary of the average relative size (in terms of number of vertices) achieved by each of the configurations on the three datasets. The best value per dataset is given in bold.

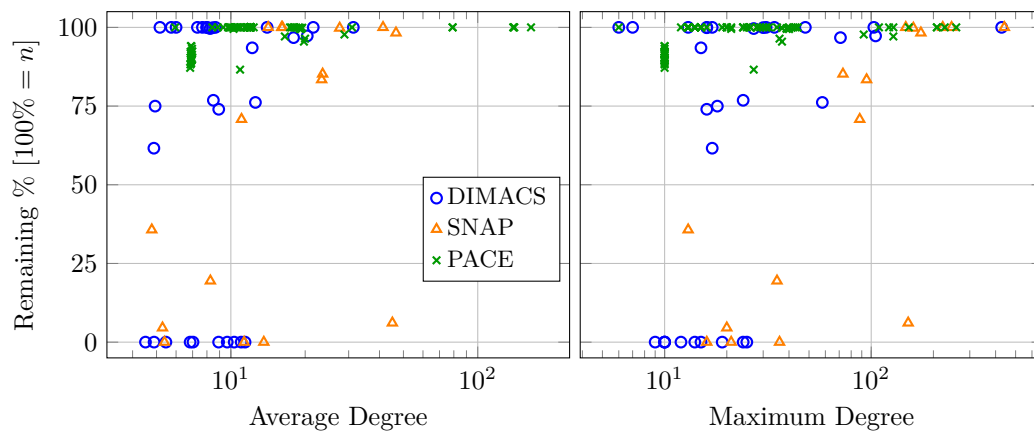
	FAR2	FAR3	ID10	ID20	ID50	LID20
DIMACS	82.6%	67.0%	62.9%	63.6%	66.1%	63.4%
SNAP	80.6%	66.8%	56.4%	59.3%	64.1%	56.4%
PACE	99.2%	97.4%	97.8%	98.1%	98.1%	98.0%

On average the ID10 configuration produced the smallest irreducible instances, see Table 2. From the FAR2 configuration it can be seen that already applying only two forward/backward rules in a sequence can considerably reduce the size of some graphs. In this case, the first rule is always a backward rule, and the second always a forward rule. However, using up to three rules in a sequence gave significantly better results.

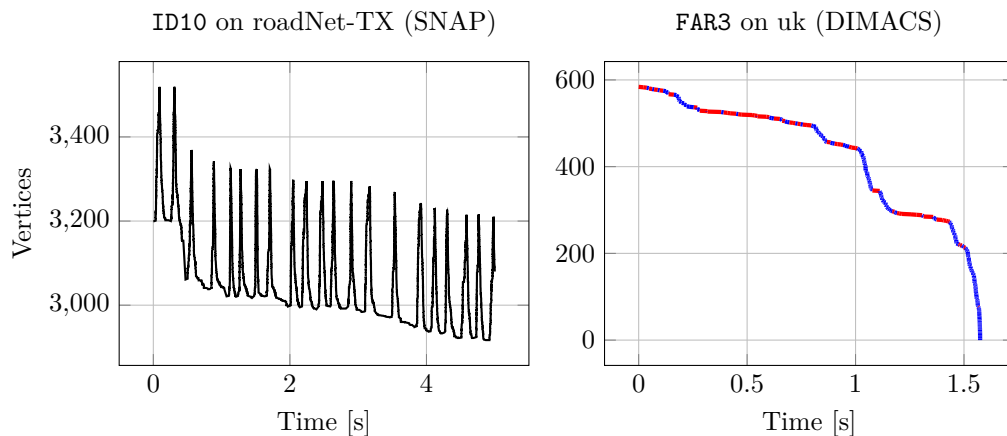
For Inflate-Deflate, we see that small inflation ratios α around 10% perform best on average. Increasing the inflation ratio leads to slightly worse results, especially for the SNAP instances.

Next, in Figure 5, we see that the ID10 method was able to shrink graphs to empty graphs mostly for graphs with the lowest average degree (8–14), with the exception of one instance with an average degree of around 45. However, a large number of graphs with an average degree of 8–14 were not shrunk considerably. The other configurations, namely ID20, ID50, and FAR3 exhibit the same behavior. Similar behavior is also observed by replacing the average degree with the maximum degree. For the most part, only graphs with a relatively small maximum degree were able to be shrunk considerably. We conclude that our approach is most viable on sparse irreducible graphs.

In Figure 6, we show how the graph size changes over time with Find and Reduce and Inflate-Deflate on two example graphs. It can be clearly observed how ID10 repeatedly increases the number of vertices, which is the inflation phase, and then subsequently reduces it, which is the deflation phase. A slow downward trend of the number of vertices can be observed.



■ **Figure 5** Scatter plot relating the relative size (measured in vertices) of the shrunk instances to the average and maximum degree of these graphs for the ID10 configuration. Note that a logarithmic scale is used for the x-axis.



■ **Figure 6** The shrinkage of two instances by ID10 and FAR3 configurations on already kernelized instances. The x axis is time, and y is number of vertices. In the right figure red segments mean that the graph was shrunk by using a sequence of at least two forward and backward rules, whereas the blue segments indicate the use of forward rules only.

For the FAR3 configuration it can be seen that there are phases in which only forward rules have to be used to shrink the graph, and phases where longer sequences of forward and backward rules are needed. Sometimes a sudden large decrease in the number of vertices using only forward rules can be observed, which one may think of as a cascading effect. The graph only had to be changed by a small amount, triggering a cascade of forward rules.

5 Conclusion

Our work showed the large potential in the general idea of undoing data reduction rules to further shrink instances that are irreducible with respect to these rules. While the results for some instances are very promising, our experiments also revealed that other instances resist our attempts of shrinking them through preprocessing. From a theory point of view this is no surprise, as we deal with NP-hard problems after all. However, there is a lot of work

that still can be done in this direction. Similar to the branch&bound approach, many clever heuristic tricks will be needed to find solutions in the vast search space. Such heuristics might, for example, employ machine learning to guide the search. As mentioned before, our approach is not limited to VERTEX COVER. Looking at other problems is future work though. A framework for applying our approach on graph problems could be another next step. Also, our approach should be easily parallelizable.

Besides all these practical questions, there are also clear theoretical challenges: For example, for a given set of data reduction rules is there *always* (for each possible instance) a sequence of backwards and forward rules to obtain an equivalent instance of constant size? Note that this would not contradict the NP-hardness of the problems: Such sequences are probably hard to find and could even be of exponential length.

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 6th Workshop on Algorithm Engineering and Experiments and the 1st Workshop on Analytic Algorithmics and Combinatorics*, pages 62–69. SIAM, 2004.
- 2 Faisal N. Abu-Khzam, Sebastian Lamm, Matthias Mnich, Alexander Noe, Christian Schulz, and Darren Strash. Recent advances in practical data reduction. *CoRR*, abs/2012.12594, 2020. [arXiv:2012.12594](https://arxiv.org/abs/2012.12594).
- 3 Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016. [doi:10.1016/j.tcs.2015.09.023](https://doi.org/10.1016/j.tcs.2015.09.023).
- 4 Gabriela Alexe, Peter L. Hammer, Vadim V. Lozin, and Dominique de Werra. Struction revisited. *Discrete Applied Mathematics*, 132(1-3):27–46, 2003. [doi:10.1016/S0166-218X\(03\)00388-3](https://doi.org/10.1016/S0166-218X(03)00388-3).
- 5 David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors. *Graph Partitioning and Graph Clustering, 10th DIMACS Implementation Challenge Workshop*, volume 588 of *Contemporary Mathematics*. American Mathematical Society, 2013. [doi:10.1090/conm/588](https://doi.org/10.1090/conm/588).
- 6 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. Pace2019: Track 1 - vertex cover instances, July 2019. [doi:10.5281/zenodo.3368306](https://doi.org/10.5281/zenodo.3368306).
- 7 M. Ayaz Dzulfikar, Johannes Klaus Fichte, and Markus Hecher. The PACE 2019 parameterized algorithms and computational experiments challenge: The fourth iteration (invited paper). In *Proceedings of the 14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *LIPICs*, pages 25:1–25:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. [doi:10.4230/LIPICs.IPEC.2019.25](https://doi.org/10.4230/LIPICs.IPEC.2019.25).
- 8 Hartmut Ehrig, Claudia Ermel, Falk Hüffner, Rolf Niedermeier, and Olga Runge. Confluence in data reduction: Bridging graph transformation and kernelization. *Computability*, 2(1):31–49, 2013. [doi:10.3233/COM-13016](https://doi.org/10.3233/COM-13016).
- 9 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *LNCS*, pages 330–356. Springer, 2018. [doi:10.1007/978-3-319-98355-4_19](https://doi.org/10.1007/978-3-319-98355-4_19).
- 10 Aleksander Figiel, Vincent Froese, André Nichterlein, and Rolf Niedermeier. There and back again: On applying data reduction rules by undoing others, 2022. [doi:10.48550/ARXIV.2206.14698](https://doi.org/10.48550/ARXIV.2206.14698).
- 11 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. [doi:10.1017/9781107415157](https://doi.org/10.1017/9781107415157).

- 12 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
- 13 Alexander Gellner, Sebastian Lamm, Christian Schulz, Darren Strash, and Bogdán Zaválnij. Boosting data reduction for the maximum weight independent set problem using increasing transformations. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX 2021)*, pages 128–142. SIAM, 2021. doi:10.1137/1.9781611976472.10.
- 14 Demian Hesse, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered: The Winning Solver from the PACE 2019 Challenge, Vertex Cover Track. In *Proceedings of the SIAM Workshop on Combinatorial Scientific Computing, CSC 2020*, pages 1–11. SIAM, 2020. doi:10.1137/1.9781611976229.1.
- 15 Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 16 Tomohiro Koana, Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche. Data reduction for maximum matching on real-world graphs: Theory and experiments. *ACM Journal of Experimental Algorithmics*, 26, 2021. doi:10.1145/3439801.
- 17 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. <https://snap.stanford.edu/data>, June 2014.
- 18 Karsten Weihe. Covering trains by stations or the power of data reduction. In *Proceedings 1st Conference on Algorithms and Experiments (ALEX98)*, pages 1–8, February 1998.

Improved Search of Relevant Points for Nearest-Neighbor Classification

Alejandro Flores-Velazco   

Department of Computer Science, University of Maryland, College Park, MD, USA

Abstract

Given a training set $P \subset \mathbb{R}^d$, the *nearest-neighbor classifier* assigns any query point $q \in \mathbb{R}^d$ to the class of its closest point in P . To answer these classification queries, some training points are more relevant than others. We say a training point is *relevant* if its omission from the training set could induce the misclassification of some query point in \mathbb{R}^d . These relevant points are commonly known as *border points*, as they define the boundaries of the Voronoi diagram of P that separate points of different classes. Being able to compute this set of points efficiently is crucial to reduce the size of the training set without affecting the accuracy of the nearest-neighbor classifier.

Improving over a decades-long result by Clarkson (FOCS'94), Eppstein (SOSA'22) recently proposed an *output-sensitive* algorithm to find the set of border points of P in $\mathcal{O}(n^2 + nk^2)$ time, where k is the size of such set. In this paper, we improve this algorithm to have time complexity equal to $\mathcal{O}(nk^2)$ by proving that the first phase of their algorithm, which requires $\mathcal{O}(n^2)$ time, are unnecessary.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases nearest-neighbor classification, nearest-neighbor rule, decision boundaries, border points, relevant points

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.54

Acknowledgements Thanks to Prof. David Mount for pointing out Eppstein's paper [12] and for the valuable discussions on the results presented in this paper.

1 Introduction

In the context of non-parametric classification, we are given a training set $P \subset \mathbb{R}^d$ consisting of n *labeled* points in d -dimensional Euclidean space, where the label of every point in P indicates the *class* (or *color*) that the point belongs to. The goal of a classifier is to use the training set P to *predict* the class for any *unlabeled* query point $q \in \mathbb{R}^d$, that is, to *classify* q .

The *nearest-neighbor classifier* (also known as *nearest-neighbor rule*) [13] stands out as a simple yet powerful method, that works by assigning any query point q to the class of its closest point in P . Despite its simplicity, the nearest-neighbor classifier is well-known to exhibit good classification accuracy both experimentally and theoretically [10, 11, 30]. In fact, it is still frequently used in many applications [5, 18, 22, 25–27, 29] over more recent and sophisticated techniques like support-vector machines [9] and deep neural networks [28].

One of the principal disadvantages of this technique is its high dependency on the size and dimensionality of the data, especially in light of *big data* applications. With training sets with billions of points becoming increasingly common, reducing the nearest-neighbor classifier's dependency on n and d is one approach to enhance its efficiency. There has been significant progress towards this goal, mainly focusing on two directions. The first involves the design of efficient data structures to answer approximate nearest-neighbor queries [2–4, 17, 19, 20, 23]. The second direction focuses on reducing the size of the training set used by the nearest-neighbor classifier, thus effectively reducing n . However, most practical techniques for training set reduction provide limited guarantees on the effect of this reduction to the accuracy of the nearest-neighbor classifier [1, 14–16].

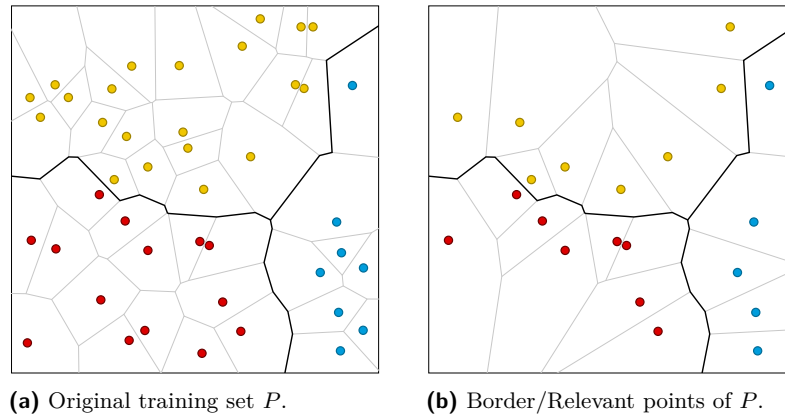


© Alejandro Flores-Velazco;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 54; pp. 54:1–54:10
Leibniz International Proceedings in Informatics



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



■ **Figure 1** On the left, a training set P with points of three classes: *red*, *blue* and *yellow*. There the black lines highlight the *boundaries* of P between points of different classes. On the right, a subset of these points corresponding to the set of border points of P . Note that by definition, the boundaries between points of different classes remain the same for P and for its set of border points.

Only a handful of works [6, 8, 12] have proposed training set reduction algorithms that guarantee the same classification of every query point, before and after the reduction took place. These are called *boundary preserving* algorithms, and it is the focus of this paper.

The set of *border points* (or *relevant points*¹) of the training set P are those that define the boundaries between points of different classes, and whose omission from the training set would imply the misclassification of some query points in \mathbb{R}^d . Formally, two points $p, \hat{p} \in P$ are border points of P if they belong to different classes, and there exist some point $q \in \mathbb{R}^d$ such that q is equidistant to both p and \hat{p} , and no other point of P is closer to q than these two points (*i.e.*, the empty ball property of Voronoi Diagrams). See Figure 1 for an example of a training set P in \mathbb{R}^2 and its set of border points. Throughout, we let k denote the total number of border points in the training set. By definition, if instead of building the nearest-neighbor classifier with the entire training set P we use the set of border points of P , its dependency is reduced from n to k , while still obtaining the same classification for any query point in \mathbb{R}^d . This becomes particularly relevant for applications where $k \ll n$.

In this paper, we improve a recently proposed algorithm by [12] that computes the set of border points of any training set $P \subset \mathbb{R}^d$, where dimension d is assumed to be constant. While the original algorithm computes such set in $\mathcal{O}(n^2 + nk^2)$ time, where k is the number of border points of P , our new algorithm computes the same set in $\mathcal{O}(nk^2)$ time.

1.1 Previous Work

Other related problems in the realm of training set reduction are NP-hard [24, 31, 33] to solve exactly (*e.g.*, those of finding minimum cardinality *consistent* subsets and *selective* subsets). However, the problem of preserving the class boundaries of the nearest-neighbor classifier, or simply, finding the set of border points of P , is tractable.

For training sets $P \subset \mathbb{R}^2$ in 2-dimensional Euclidean space, Bremner *et al.* [6] proposed an output-sensitive algorithm for finding the set of border points of P in $\mathcal{O}(n \log k)$ worst-case time. However, how to generalize this algorithm for higher dimensions remained unclear.

¹ While [12] uses the term *relevant points*, the term *border points* has been the standard in the literature of this and other related problems [14, 15, 21, 32]. For this reason, we stick to the term *border points*.

Until very recently, the best result for the higher dimensional case was that of Clarkson [8]. He proposed an algorithm to find the set of border points of $P \subset \mathbb{R}^d$, with bounded d , that runs in $\mathcal{O}(\min(n^3, kn^2 \log n))$ worst-case time. For almost three decades, this remained the best result for training sets in \mathbb{R}^d . Recently, Eppstein [12] proposed a significantly faster algorithm for the d -dimensional Euclidean case, which runs in $\mathcal{O}(n^2 + nk^2)$ worst-case time.

Eppstein’s algorithm is strikingly simple, yet full of interesting ideas (see Algorithm 1). The algorithm works as follows: it begins by selecting an initial set of border points of P , one point from every class region. From here, the algorithm uses a series of subroutines which we will group together and denote as the “*inversion method*”, to find the remaining border points of P . Thus, the algorithm can be naturally split into two phases: the initialization of R with some border points, and the search process for the remaining border points of P .

■ **Algorithm 1** Recent Eppstein’s algorithm [12] to find the set of border points of P .

Input: Initial training set P
Output: The set of border points of P

- 1 Let M be the MST of P
- 2 Initialize R with the end points of every bichromatic edge of M
- 3 **foreach** $p \in R$ **do**
- 4 Let c be p ’s class and P_c be the points of P that belong to class c
- 5 Let S_p be the inverted points of $P \setminus P_c$ around p
- 6 Find all extreme points of S_p and their corresponding original points E_p
- 7 $R \leftarrow R \cup E_p$
- 8 **return** R

The initialization phase (lines 1–2 of Algorithm 1) involves finding a subset of border points such that at least one point for every class region is selected. Eppstein observes that this can be achieved by computing the Minimum Spanning Tree (MST) of P , identifying the edges of the MST that connect points of different classes (denoted as *bichromatic* edges), and selecting the endpoints of all such edges. This phase takes $\mathcal{O}(n^2)$ time, but we will prove that it is not necessary.

The search phase (lines 3–6 of Algorithm 1) is in charge of finding every remaining border point of P . This phase iterates over all selected points, and for each such point p , it performs what we call the inversion method. This method identifies a subset of border points of P , which are added to R . Once the algorithm has done the inversion method on every point of R , it terminates with the guarantee of having selected every border point of P .

Given any point $p \in P$, the inversion method on p is described in lines 4–6 of Algorithm 1. Let c be p ’s class, and P_c be the points of P that belong to class c , the inversion method on p consists of: (i) inverting all points of $P \setminus P_c$ around a ball centered at p (call the set of these inverted points as S_p and include p itself in the set), (ii) computing the set of extreme points of S_p , and finally (iii) returning the set E_p of those points of P that correspond to the extreme points of S_p before inversion. For a detailed description and proof of correctness of this method, we refer the reader to Eppstein’s paper [12]. However, for the purposes of this paper we only need a property presented in Lemma 3 of [12]: the points in E_p reported by the inversion method are the Delaunay neighbors of p with respect to the set $(P \setminus P_c) \cup \{p\}$.

Every call of the inversion method takes $\mathcal{O}(nk)$ time by leveraging well-known output-sensitive algorithms for computing extreme points. Given that this method is called exclusively on every border point of the training set, this yields a total of $\mathcal{O}(nk^2)$ time to complete the search phase of the algorithm. Overall, this implies that Eppstein’s algorithm computes the entire set of border points of P in $\mathcal{O}(n^2 + nk^2)$ worst-case time.

2 Our Approach

We propose a simple modification to Eppstein’s algorithm, which avoids the step of computing the MST of the training set P , along with the subsequent selection of bichromatic edges to produce the initial subset of border points.

Instead, we simply start the search process with any arbitrary point of P . The rest of the algorithm remains virtually unchanged (see Algorithm 2 for a formal description). We show that this new approach is not only correct, meaning that it only finds border points of P , but also complete, as all border points of P are eventually found by our algorithm. Additionally, by avoiding the main bottleneck of the original algorithm, our new algorithm computes the same result in $\mathcal{O}(nk^2)$ time, eliminating the $\mathcal{O}(n^2)$ term.

■ **Algorithm 2** New algorithm to find the set of border points of P .

Input: Initial training set P
Output: The set of border points of P

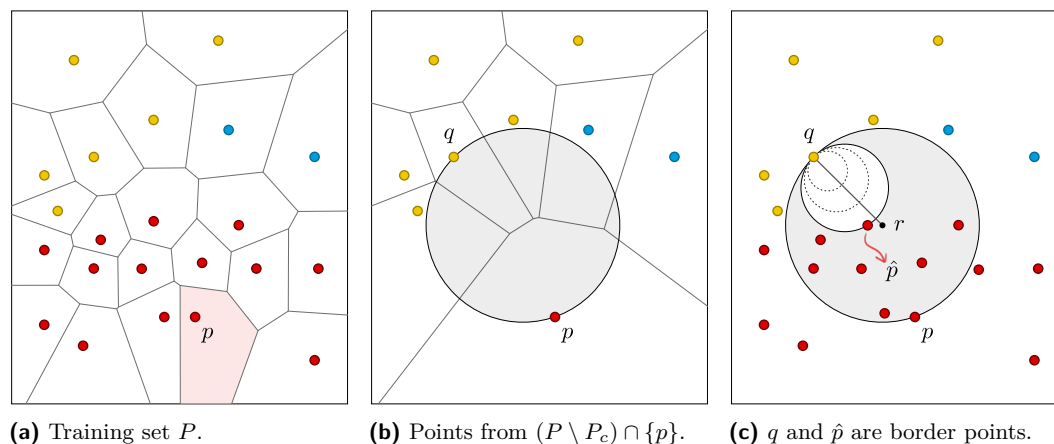
- 1 Let s be any “seed” point from P
- 2 $R \leftarrow \phi$
- 3 **foreach** $p \in R \cup \{s\}$ **do**
- 4 Let c be p ’s class and P_c be the points of P that belong to class c
- 5 Let S_p be the inverted points of $P \setminus P_c$ around p
- 6 Find all extreme points of S_p and their corresponding original points E_p
- 7 $R \leftarrow R \cup E_p$
- 8 **return** R

Before proceeding, it is useful to explore why Eppstein’s algorithm computes the MST of the training set P . First, note that the original algorithm only applies the inversion method on border points of P . In fact, Eppstein’s correctness proof relies on it: Lemma 6 in [12] proves that all points in E_p are border points by assuming that point p is also a border point. From the description of our algorithm, note that we initially apply the inversion method on a “seed” point s , which might not be a border point. Therefore, we need to generalize Lemma 6 in [12] for the case where p is not a border point of P . Additionally, using the points from all bichromatic pairs of the MST of P guarantees that Eppstein’s algorithm starts the search phase with at least one point from every boundary of P . Eppstein’s completeness proof shows that this search can then “move along” any given boundary and eventually select all its defining points. We show that the search process is far more powerful, and can even “jump” between nearby boundaries, thus rendering the MST computation unnecessary.

The following description outlines the necessary steps to prove both the correctness and completeness of our new algorithm, which are unfolded in the rest of this section.

- By applying the inversion method to any point of P , not necessarily a border point, all reported points are border points of P . This is established in Lemma 1, generalizing the statement of Lemma 6 of [12] for non-border points.
- For any class boundary of P , once the algorithm selects a point from this boundary, it will eventually select every other point defining the same boundary. This is originally proved in Lemma 10 [12], however, we provide simpler proofs in Lemmas 2 and 3.
- Given two disconnected boundaries separated by a class region, we prove that if our algorithm selects a defining point from one of the boundaries, it will eventually select all defining points from both boundaries. This is proved in Lemma 4.

All together, these lemmas are used to prove the main result: the correctness, completeness, and worst-case time complexity of Algorithm 2, as stated in Theorems 5 and 6.



■ **Figure 2** Example showing the inversion method from any point $p \in P$. On the left, training set P . The middle figure shows every non red point of P , except for p itself, along with a point q selected from the inversion method on p . On the right, we see evidence that q is a border point of P .

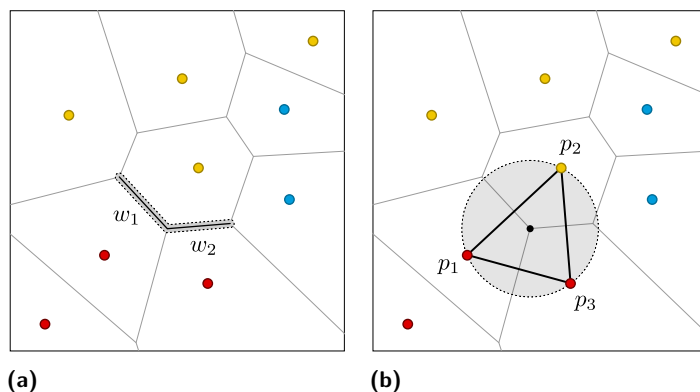
► **Lemma 1.** *Let $p \in P$ be any point of the training set. Then every point selected using the inversion method on p must be a border point of P .*

Proof. Let E_p be the points of P corresponding (before inversion) to the extreme points of S_p . According to Lemma 3 [12], every point in E_p is a neighbor of point p with respect to the Voronoi Diagram of set $(P \setminus P_c) \cup \{p\}$. This implies that for every point $q \in E_p$ other than p , there exists a ball such that both p and q are on its surface and no points of $P \setminus P_c$ lie inside (see Figures 2a and 2b). We can now leverage similar techniques to the ones described in [6], to find a “witness” point to the hypothesis that q must be a border point of P .

Recall that the empty ball we just described, as illustrated in Figure 2b, is empty from points of $P \setminus P_c$. However, there might be points of P_c inside. And moreover, we know that at least one point of P_c , point p , lies on its surface. Now, let r be the center of this ball, we grow an empty ball, this time with respect to the entire training set P , such that its center lies on the line \overline{qr} and point q is on its surface (see Figure 2c). This ball will grow until it hits another point \hat{p} of P , which we are guaranteed it will be of the same class as point p , and thus, of different class as point q . Finally, we have just found an empty ball with respect to P , which has points q and \hat{p} on its surface, and were the class of both points differ. Therefore, this implies that q is a border point of P . ◀

Before continuing, we need to formally define a few concepts. First, we define a *wall* of P as any $(d - 1)$ -dimensional face of the Voronoi Diagram of P . By known properties of these structures, every wall w is *defined* by two distinct points $p, q \in P$ such that any point on w has p and q as its two equidistant nearest-neighbors in the training set. We say two walls are *adjacent* if their intersection is not empty. That is, if there exists a point in \mathbb{R}^d with all the defining points of these two walls as its equidistant nearest-neighbors in P .

Additionally, we define a *class boundary* (or just *boundary*) of P as the union of adjacent walls, where each of these walls is defined by two points of different classes. Similarly, we define a *class region* of P as the union of adjacent Voronoi cells whose defining points belong to the same class. Based on these definitions, note that class boundaries are the ones that separate different class regions of P . Figure 4 illustrates a training set in \mathbb{R}^2 with points of three classes, whose Voronoi Diagram describes five class regions and two class boundaries.

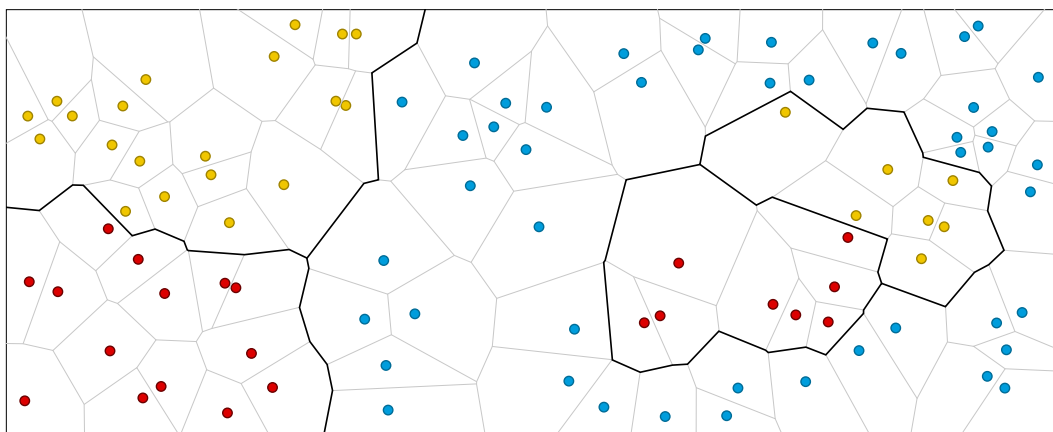


■ **Figure 3** By definition, any two adjacent walls w_1 and w_2 of the Voronoi Diagram of P hold the empty ball property with the points that define them. When these walls are part of the class boundaries of P , the points that define them belong to at least two classes.

► **Lemma 2.** Let w_1 and w_2 be two adjacent walls in a class boundary of P . If the algorithm selects one of the points defining one of these walls, it eventually selects the remaining points defining both walls.

Proof. Let \mathcal{W} be the set of points defining both walls w_1 and w_2 (see Figure 3). By definition, these two walls of the Voronoi Diagram of P are adjacent if there exists an empty ball with all the points of \mathcal{W} on its surface. Knowing these two walls are part of the class boundaries of P , the set \mathcal{W} must contain at least three points, and at least two classes.

Let p_1 be the first point of \mathcal{W} to be selected by the algorithm. When doing the inversion method on point p_1 , the algorithm will select all points of \mathcal{W} of different class than p_1 , of which we know there is at least one. Let p_2 be one such point. Finally, when doing the inversion method on point p_2 , the algorithm will select the remaining points of \mathcal{W} of the same class as p_1 . Therefore, all points of \mathcal{W} will eventually be selected by the algorithm. ◀



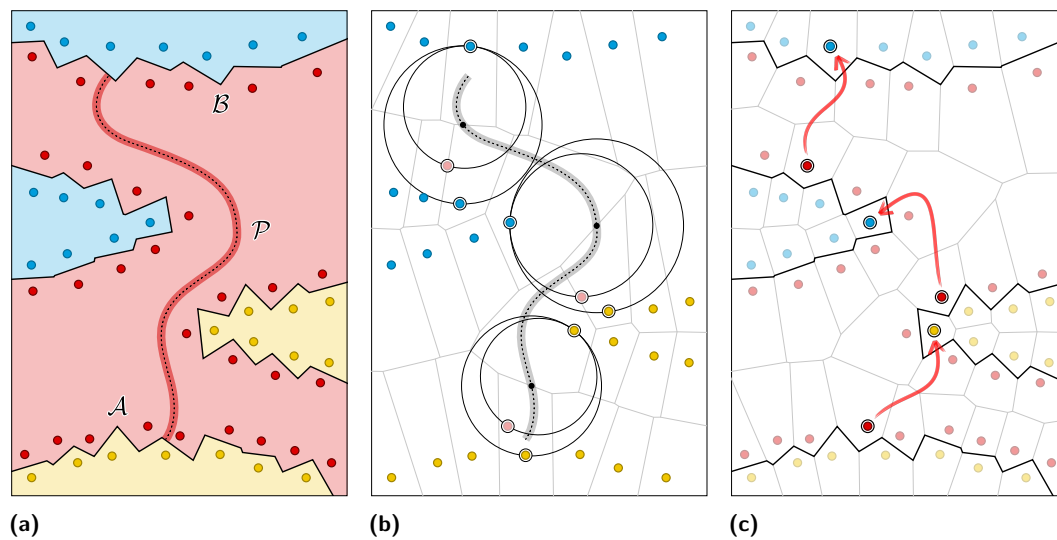
■ **Figure 4** A training set with five class regions (one blue, two red, and two yellow regions), along with two disconnected class boundaries that separate all these regions. On the left, a boundary separating the blue region and the leftmost yellow and red regions. On the right, a boundary that separates the same blue region and the remaining red and yellow regions.

► **Lemma 3.** *Let \mathcal{A} be a class boundary of P , and assume that the algorithm selects one of the defining points of \mathcal{A} . Then, the algorithm will eventually select all defining points of \mathcal{A} .*

This comes as a direct consequence of Lemma 2 and the definition of a class boundary of the training set P . It remains to show what happens with boundaries that are disconnected.

► **Lemma 4.** *Let \mathcal{A} and \mathcal{B} be two disconnected boundaries of P , such that there exists a path in space from \mathcal{A} to \mathcal{B} that is completely contained within one color region. Without loss of generality, say that every point that defines \mathcal{A} has been selected by the algorithm. Then, every point that defines \mathcal{B} must also be selected by the algorithm.*

Proof. Given these two disconnected boundaries \mathcal{A} and \mathcal{B} , we assume there exists some path \mathcal{P} in \mathbb{R}^d going from a wall of \mathcal{A} to a wall of \mathcal{B} , such that this path passes exclusively through a single class region (see Figure 5a). Without loss of generality, say this is a *red* class region. Formally, for every point r along \mathcal{P} we know r 's nearest-neighbor in P is red. Additionally, we assume that every border point defining \mathcal{A} is selected by the algorithm. Hence, the proof consists of showing that there exists a sequence of border points $\langle p_1, \hat{p}_1, p_2, \hat{p}_2, \dots, p_m, \hat{p}_m \rangle$ such that (i) p_1 and \hat{p}_m are defining points of \mathcal{A} and \mathcal{B} , respectively, (ii) \hat{p}_i is retrieved by the inversion method on p_i , for every $i \in [1, m]$, and finally (iii) points p_i and \hat{p}_{i-1} are both defining the same boundary, for every $i \in [2, m]$. See Figure 5 for a visual description.



■ **Figure 5** On the right, two disconnected boundaries \mathcal{A} and \mathcal{B} enclosing a red class region. Thus, there is a path \mathcal{P} completely contained inside such region and connecting both boundaries. Other boundaries can also be enclosing the same region and be near path \mathcal{P} . On the left, we prove that there exists a sequence of points that can be retrieved by calls to the inversion method, such that if points of \mathcal{A} are selected by the algorithm, eventually points of \mathcal{B} will also be selected.

By definition, for every point r along path \mathcal{P} we know r 's nearest-neighbor is a red point. Now, let's delete every red point from consideration, including the ones defining boundaries \mathcal{A} and \mathcal{B} (see Figure 5b). This immediately implies that r 's nearest-neighbor just became a non-red border point of P . The fact that r 's new nearest-neighbor is a border point is easy to prove, using similar arguments as the ones laid down in Lemma 1. Additionally, these border points could be defining other boundaries apart from \mathcal{A} and \mathcal{B} , as seen in Figure 5b.

Let's start moving along the path \mathcal{P} , starting from the end-point of the path that lies on a wall of boundary \mathcal{A} . Then, find all r_i points along the path, where each r_i has two equidistant nearest-neighbors among the remaining non-red points, and both points define

two distinct boundaries of P . We say there are m of these points along the path, and denote r_i 's two equidistant nearest-neighbors as $q_{i,1}$ and $q_{i,2}$ for $i \in [1, m]$. Clearly, $q_{i,1}$ and $q_{i-1,2}$ are border points defining the same boundary, for all $i \in [2, m]$. See Figure 5b, where the three black points along the path are the r_i points, and the *yellow* and *blue* points on the surface of the balls centered at each r_i are the corresponding $q_{i,1}$ and $q_{i,2}$ points.

For now, let's fix the analysis on one such r_i point, and consider the ball centered at r_i with both $q_{i,1}$ and $q_{i,2}$ on its surface. There must exist some other point $q_{i,3}$ lying inside of r_i 's ball, such that $q_{i,3}$ is one of the deleted red points defining the same boundary as $q_{i,1}$. It is now easy to see that there exist an empty ball, with respect to the set $P \setminus P_{red} \cup \{q_{i,3}\}$, with both $q_{i,3}$ and $q_{i,2}$ on its boundary. This implies that $q_{i,2}$ is retrieved by the inversion method on $q_{i,3}$. Therefore, let's add $p_i \leftarrow q_{i,3}$ and $\hat{p}_i \leftarrow q_{i,2}$ to the sequence of points that we are looking for. Repeat this for every r_i with $i \in [1, m]$ to identify all points in the sequence.

Finally, we have the sequence of border points $\langle p_1, \hat{p}_1, p_2, \hat{p}_2, \dots, p_m, \hat{p}_m \rangle$ such that for any $i \in [1, m]$ assuming that the algorithm selects the points defining the same boundary as p_i , it will also select \hat{p}_i , and leveraging Lemma 3 it will eventually select all other points defining the same boundary as \hat{p}_i . Given that p_1 and \hat{p}_m are defining border points of boundaries \mathcal{A} and \mathcal{B} , respectively, and by the assumption that all points defining \mathcal{A} are selected by the algorithm, we know that eventually, all points defining \mathcal{B} will be selected too. ◀

► **Theorem 5.** *The algorithm selects every border point of P in $\mathcal{O}(nk^2)$ time.*

Proof. Proving the worst-case time complexity of our algorithm follows directly from the time complexity of the search phase of Eppstein's algorithm [12]. However, the correctness and completeness of our algorithm follows from Lemmas 1 to 4.

First, we know by Lemmas 1-3 that Algorithm 2 will select the defining border points of at least one class boundary of P . Denote this boundary as \mathcal{A} and consider any other boundary \mathcal{B} of P . Evidently, we can draw a path \mathcal{P} from \mathcal{A} to \mathcal{B} , which would generally pass through several class regions. Then, let's split \mathcal{P} into several subpaths $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m$ such that each subpath is completely contained within a single class region. From this, we can directly apply Lemma 4 on each of the intermediate boundaries that "cut" \mathcal{P} into these subpaths. Finally, this implies that our algorithm will eventually select every defining point of boundary \mathcal{B} , and similarly, it will do the same with all other boundaries of P . ◀

► **Theorem 6.** *Leveraging Chan's algorithm [7] for finding extreme points, the algorithm selects every border point of P in randomized expected time $\mathcal{O}(nk \log k)$ for $d = 3$, and in*

$$\mathcal{O}\left(nk(\log k)^{\mathcal{O}(1)} + k(nk)^{1 - \frac{1}{\lfloor d/2 \rfloor + 1}} (\log n)^{\mathcal{O}(1)}\right)$$

time for all constant dimensions $d > 3$.

Just as with Eppstein's original algorithm, we can use Chan's randomized algorithm [7] for finding extreme points of point sets in \mathbb{R}^d , in order to reduce the expected time complexity of our improved algorithm. The remaining of the proof is the same as for Theorem 5.


References

- 1 Fabrizio Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464, 2007.
- 2 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 Sunil Arya, Guilherme D. Da Fonseca, and David M. Mount. Approximate polytope membership queries. *SIAM Journal on Computing*, 47(1):1–51, 2018.
- 4 Sunil Arya, Theocharis Malamatos, and David M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *Journal of the ACM (JACM)*, 57(1):1, 2009.
- 5 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.
- 6 David Bremner, Erik Demaine, Jeff Erickson, John Iacono, Stefan Langerman, Pat Morin, and Godfried Toussaint. Output-sensitive algorithms for computing nearest-neighbour decision boundaries. In Frank Dehne, Jörg-Rüdiger Sack, and Michiel Smid, editors, *Algorithms and Data Structures: 8th International Workshop, WADS 2003, Ottawa, Ontario, Canada, July 30 - August 1, 2003. Proceedings*, pages 451–461, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/978-3-540-45078-8_39.
- 7 Timothy M Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete & Computational Geometry*, 16(4):369–387, 1996.
- 8 Kenneth L Clarkson. More output-sensitive geometric algorithms. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 695–702. IEEE, 1994.
- 9 Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.
- 10 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.
- 11 Luc Devroye. On the inequality of cover and hart in nearest neighbor discrimination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, pages 75–78, 1981.
- 12 David Eppstein. Finding relevant points for nearest-neighbor classification. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 68–78. SIAM, 2022.
- 13 E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine*, Technical Report 4(3):477+, January 1951.
- 14 Alejandro Flores-Velazco. Social distancing is good for points too! In *Proceedings of the 32nd Canadian Conference on Computational Geometry, CCCG 2020, August 5-7, 2020, University of Saskatchewan, Saskatoon, Saskatchewan, Canada, 2020*.
- 15 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*.
- 16 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.
- 17 Alejandro Flores-Velazco and David M. Mount. Boundary-sensitive approach for approximate nearest-neighbor classification. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 44:1–44:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.44.
- 18 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. arXiv:1908.10396.
- 19 Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization, 2020. arXiv:1908.10396.
- 20 Sariel Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.

- 21 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.
- 22 Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *arXiv preprint*, 2017. [arXiv:1702.08734](https://arxiv.org/abs/1702.08734).
- 23 Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus, 2017. [arXiv:1702.08734](https://arxiv.org/abs/1702.08734).
- 24 Kamyar Khodamoradi, Ramesh Krishnamurti, and Bodhayan Roy. Consistent subset problem with two labels. In *Conference on Algorithms and Discrete Applied Mathematics*, pages 131–142. Springer, 2018.
- 25 Marc Houry and Dylan Hadfield-Menell. Adversarial training with Voronoi constraints. *CoRR*, abs/1905.01019, 2019. [arXiv:1905.01019](https://arxiv.org/abs/1905.01019).
- 26 Nicolas Papernot and Patrick McDaniel. Deep k -nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint*, 2018. [arXiv:1803.04765](https://arxiv.org/abs/1803.04765).
- 27 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.
- 28 Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014. [arXiv:1404.7828](https://arxiv.org/abs/1404.7828).
- 29 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.
- 30 Charles J. Stone. Consistent nonparametric regression. *The annals of statistics*, pages 595–620, 1977.
- 31 Gordon Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, pages 224–233, New York, NY, USA, 1991. ACM. [doi:10.1145/109648.109673](https://doi.org/10.1145/109648.109673).
- 32 D. Randall Wilson and Tony R. Martinez. Instance pruning techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 403–411, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. URL: <http://dl.acm.org/citation.cfm?id=645526.657143>.
- 33 A. V. Zuhba. NP-completeness of the problem of prototype selection in the nearest neighbor method. *Pattern Recog. Image Anal.*, 20(4):484–494, December 2010. [doi:10.1134/S1054661810040097](https://doi.org/10.1134/S1054661810040097).



Longest Cycle Above Erdős–Gallai Bound

Fedor V. Fomin  

Department of Informatics, University of Bergen, Norway

Petr A. Golovach  

Department of Informatics, University of Bergen, Norway

Danil Sagunov  

St. Petersburg Department of V.A. Steklov Institute of Mathematics, Russia

Kirill Simonov 

Algorithms and Complexity Group, TU Wien, Austria

Abstract

In 1959, Erdős and Gallai proved that every graph G with average vertex degree $\text{ad}(G) \geq 2$ contains a cycle of length at least $\text{ad}(G)$. We provide an algorithm that for $k \geq 0$ in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ decides whether a 2-connected n -vertex graph G contains a cycle of length at least $\text{ad}(G) + k$. This resolves an open problem explicitly mentioned in several papers. The main ingredients of our algorithm are new graph-theoretical results interesting on their own.

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph algorithms; Theory of computation \rightarrow Parameterized complexity and exact algorithms

Keywords and phrases Longest path, longest cycle, fixed-parameter tractability, above guarantee parameterization, average degree, Erdős and Gallai theorem

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.55

Related Version *Full Version:* <https://arxiv.org/abs/2202.03061>

Funding *Fedor V. Fomin:* supported by the Research Council of Norway via the project BWCA (grant no. 314528).

Petr A. Golovach: supported by the Research Council of Norway via the project BWCA (grant no. 314528).

Danil Sagunov: supported by Leonhard Euler International Mathematical Institute in Saint Petersburg (agreement no. 075-15-2019-1620).

Kirill Simonov: supported by the Austrian Science Fund (FWF) via project Y1329 (Parameterized Analysis in Artificial Intelligence).

1 Introduction

The circumference of a graph is the length of its longest (simple) cycle. In 1959, Erdős and Gallai [4] gave the following, now classical, lower bound for the circumference of an undirected graph.

► **Theorem 1** (Erdős and Gallai [4]). *Every graph with n vertices and more than $\frac{1}{2}(n-1)\ell$ edges ($\ell \geq 2$) contains a cycle of length at least $\ell + 1$.*

We provide an algorithmic extension of the Erdős–Gallai theorem: A fixed-parameter tractable (FPT) algorithm with parameter k , that decides whether the circumference of a graph is at least $\ell + k$. To state our result formally, we need a few definitions. For an undirected graph G with n vertices and m edges, we define $\ell_{EG}(G) = \frac{2m}{n-1}$. Then by the Erdős–Gallai theorem, G always has a cycle of length at least $\ell_{EG}(G)$ if $\ell_{EG}(G) > 2$. The parameter $\ell_{EG}(G)$ is closely related to the *average degree* of G , $\text{ad}(G) = \frac{2m}{n}$. It is easy to see that for every graph G with at least two vertices, $\ell_{EG}(G) - 1 \leq \text{ad}(G) < \ell_{EG}(G)$.



© Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 55; pp. 55:1–55:15

Leibniz International Proceedings in Informatics



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

55:2 Longest Cycle Above Erdős–Gallai Bound

The *maximum average degree* $\text{mad}(G)$ is the maximum value of $\text{ad}(H)$ taken over all induced subgraphs H of G . Note that $\text{ad}(G) \leq \text{mad}(G)$ and $\text{mad}(G) - \text{ad}(G)$ may be arbitrarily large. By Goldberg [13] (see also [12]), $\text{mad}(G)$ can be computed in polynomial time. By Theorem 1, we have that if $\text{ad}(G) \geq 2$, then G has a cycle of length at least $\text{ad}(G)$ and, furthermore, if $\text{mad}(G) \geq 2$, then there is a cycle of length at least $\text{mad}(G)$. Based on this guarantee, we define the following problem.

LONGEST CYCLE ABOVE MAD

Input: A graph G on n vertices and an integer $k \geq 0$.
Task: Decide whether G contains a cycle of length at least $\text{mad}(G) + k$.

Our main result is that this problem is FPT parameterized by k . More precisely, we show the following.

► **Theorem 2.** *LONGEST CYCLE ABOVE MAD can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ on 2-connected graphs.*

While Theorem 2 concerns the decision variant of the problem, its proof may be easily adapted to produce a desired cycle if it exists. We underline this because the standard construction of a long cycle that for every $e \in E(G)$ invokes the decision algorithm on $G - e$, does not work in our case, as edge deletions decrease the average degree of a graph.

Theorem 2 has several corollaries. The following question was explicitly stated in the literature [6, 9]. For a 2-connected graph G and a nonnegative integer k , how difficult is it to decide whether G has a cycle of length at least $\text{ad}(G) + k$? According to [9], it was not known whether the problem parameterized by k is FPT, W[1]-hard, or Para-NP. Even the simplest variant of the question, whether a path of length $\text{ad}(G) + 1$ can be computed in polynomial time, was open. Theorem 2 resolves this question because $\text{mad}(G) \geq \text{ad}(G)$ for every graph G .

► **Corollary 3.** *For a 2-connected graph G and a nonnegative integer k , deciding whether G has a cycle of length at least $\text{ad}(G) + k$ can be done in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.*

Similarly, we have the following corollary.

► **Corollary 4.** *For a 2-connected graph G and a nonnegative integer k , deciding whether G has a cycle of length at least $\ell_{EG}(G) + k$ can be done in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.*

An undirected graph G is *d-degenerate* if every subgraph of G has a vertex of degree at most d , and the *degeneracy* of G is defined to be the minimum value of d for which G is d -degenerate. Since a graph of degeneracy d has a subgraph H with at least $d \cdot |V(H)|/2$ edges, we have that $d \leq \text{ad}(H) \leq \text{mad}(G)$. Therefore, Theorem 2 implies the following corollary, which is the main result of [6].

► **Corollary 5 ([6]).** *For a 2-connected graph G of degeneracy d , deciding whether G has a cycle of length at least $d + k$ can be done in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.*

Theorem 1 provides the same lower bound on the number of vertices in a longest path. We consider the LONGEST PATH ABOVE MAD problem that, given a graph G and integer k , asks whether G has a path with at least $\text{mad}(G) + k$ vertices. Observe that a graph G has a path with ℓ vertices if and only if the graph G' , obtained by adding to G a universal vertex that is adjacent to every vertex of the original graph, has a cycle with $\ell + 1$ vertices. Because $\text{mad}(G') \geq \text{mad}(G)$, Theorem 2 yields the following.

► **Corollary 6.** *LONGEST PATH ABOVE MAD can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ on connected graphs.*

We complement Theorem 2 by observing that the 2-connectivity condition is crucial for tractability due to the fact that the considered properties are not closed under taking biconnected components. In particular, it may happen that every long cycle of a graph is in a biconnected component of small average degree. This yields the following theorem.

► **Theorem 7** (★).¹ *It is NP-complete to decide whether an n -vertex connected graph G has a cycle of length at least $\ell_{EG}(G) + 1$.*

The single-exponential dependence in k of algorithm in Theorem 2 is asymptotically optimal: it is unlikely that LONGEST CYCLE ABOVE MAD can be solved in $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ time. This immediately follows from the well-known result (see e.g. [2, Chapter 14]) that existence of an algorithm for HAMILTONIAN CYCLE with running time $2^{o(n)}$ would refute the *Exponential Time Hypothesis* (ETH) of Impagliazzo, Paturi, and Zane [14]. Thus LONGEST CYCLE ABOVE MAD cannot be solved in $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ time, unless ETH fails.

Comparison with the previous work. Two of the recent articles on the circumference of a graph above guarantee are most relevant to our work. The first is the paper of Fomin, Golovach, Lokshtanov, Panolan, Saurabh, and Zehavi [6] who gave an algorithm that in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ for a 2-connected graph G of degeneracy d , decides whether G has a cycle of length at least $d + k$. In the heart of their algorithm is the following “rerouting” argument: If a cycle hits a sufficiently “dense” subgraph H of G , then this cycle can be rerouted inside H to cover all vertices of H . The main obstacle on the way of generalizing the result of Fomin et al. [6] “beyond” the average degree was the lack of rerouting arguments in graphs of large average degree.

The rerouting arguments in the proof of Theorem 2 use the structural properties of dense graphs developed in the recent work of Fomin, Golovach, Sagunov, and Simonov [9] (see [8] for the full version) on parameterized complexity of finding a cycle above Dirac’s bound. We remind that by the classical theorem of Dirac [3], every 2-connected graph has a cycle of length at least $\min\{2\delta(G), |V(G)|\}$, where $\delta(G)$ is the minimum degree of G . Fomin et al. gave an algorithm that in time $2^{\mathcal{O}(k+|B|)} \cdot n^{\mathcal{O}(1)}$ decides whether a 2-connected graph G contains a cycle of length at least $\min\{2\delta(G-B), |V(G)| - |B|\} + k$, where B is a given subset of vertices which may have “small” degrees. The result of Fomin et al. [8, 9] is “orthogonal” to ours in the following sense: It does not imply Theorem 2 and Theorem 2 does not imply the theorem from [8]. However, the tools developed in [8], in particular the new type of graph decompositions called *Dirac decompositions*, appear to be useful in our case too.

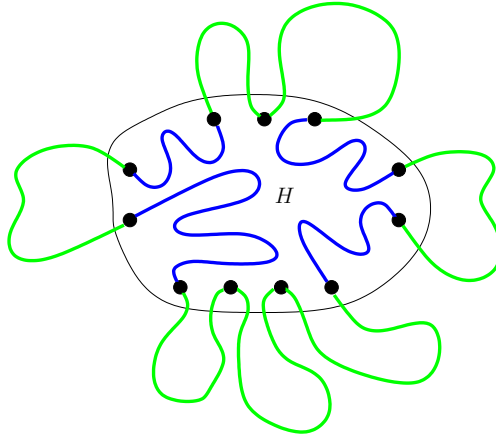
From a more general perspective, our work belongs to a popular subfield of Parameterized Complexity concerning parameterization above/below specified guarantees. In addition to [9, 6], the parameterized complexity of paths and cycles above some guarantees was studied in [1, 15], and [7].

2 Overview of the proof of the main result

Here we outline the critical technical ideas leading to our main result, Theorem 2. We first explain our techniques for the LONGEST CYCLE ABOVE AD problem. Let us remind that in this problem, the task is to decide whether a graph G has a cycle of length at least $\text{ad}(G) + k$. (The difference with *mad* is that we do not take the maximum over all subgraphs.)

¹ The results with omitted proofs are marked with the “★” sign. Missing proofs can be found in the full version of this paper [10].

The nucleus of our proof is a novel structural analysis of dense subgraphs in graphs with large average degrees. Informally, we prove that if there is a cycle of length at least $\text{ad}(G) + k$ in G , then G contains a dense subgraph H and a long (of length at least $\text{ad}(G) + k$) cycle C that “revolves” around H (see Figure 1). By that, we mean the following. First, the number of times cycle C enters and leaves H is bounded by $\mathcal{O}(k)$. Second, C contains at least $\text{ad}(G) - ck$ vertices of H for some constant c . Moreover, we need a way stronger “routing” property of H . Basically for any possible “points of entry and departure” of cycle C in H , we show that these pairs of vertices could be connected in H by internally vertex-disjoint paths of total length at least $\text{ad}(G) - ck$. Furthermore, such paths could be found in polynomial time. Then everything boils down to the following problem. For a given subgraph H of G , we are looking for at most k internally vertex-disjoint paths outside H of total length $\Omega(k)$, each path starts and ends in H . This task can be done in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ by making use of color-coding. Finally, if we find such paths, then we could complete them to a cycle of length at least $\text{ad}(G) + k$ by augmenting them by the paths inside H .



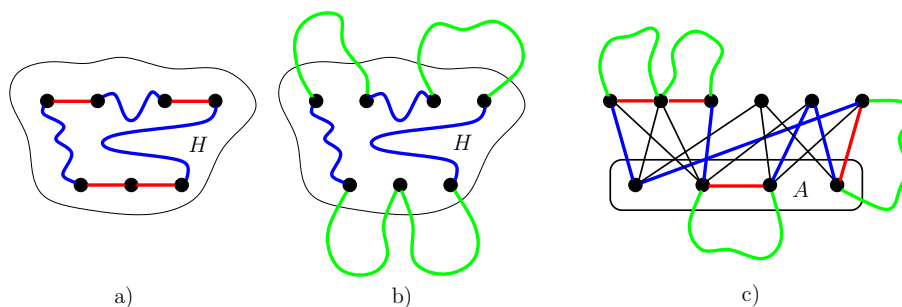
■ **Figure 1** A cycle “revolving” around H . The segments of the cycle outside H are shown in green and the segments inside H are blue.

Identifying dense subgraph H . Notice that we can assume that $\text{ad}(G) \geq \alpha k$ for a sufficiently big positive constant α . Otherwise, we can solve the problem in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time using the known algorithm for LONGEST CYCLE [11]. We start with preprocessing rules “illuminating” some “useless” parts of the graph. If G contains several connected components, it suffices to keep only the densest of them, as its average degree is at least the average degree of G . Similarly, if G is connected but has a cut-vertex, keeping the densest block also suffices. Further, if there is a vertex v of degree less than $\frac{1}{2}\text{ad}(G)$, then v can be safely removed. By applying these reduction rules exhaustively, we find an induced 2-connected subgraph H of G whose minimum degree $\delta(H) \geq \frac{1}{2}\text{ad}(H) \geq \frac{1}{2}\text{ad}(G)$. Similarly to removing sparse blocks, if G contains a vertex separator X of size two such that there is a “sparse” component A of $G - X$, then A can be removed. By applying the last reduction rule we either find a cycle of length at least $\text{ad}(G) + k$ or can conclude that the resulting subgraph H is 3-connected.

If (G, k) is a yes-instance, that is, graph G contains a cycle of length at least $\text{ad}(G) + k$, there are two possibilities. Either in G a cycle of length at least $2\delta(H) + k$ “lives” entirely in H , or it passes through some other vertices of G . If a long cycle is entirely in H , we can employ the recent result of Fomin et al. [8] that finds in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ in a 2-connected graph G a cycle of length at least $2\delta(G) + k \geq \text{ad}(G) + k$. However, if no long cycle lives entirely in H , the result of Fomin et al. is not applicable.

The next step of constructing H crucially benefits from the graph-theoretical result of Fomin et al. [8]. Specifically, we use the theorem about the Dirac decomposition from [8]. The definition of the Dirac decomposition is technical and we give it in Section 4. For 2-connected graphs, the Dirac decomposition imposes a very intricate structure. However, since, thanks to the reduction rules, H is 3-connected, we bypass most of the technical details from [8]. Informally, the Dirac decomposition leads to the following win-win situation. By the Dirac's theorem [3], graph H contains a cycle S of length at least $2\delta(H) \geq \text{ad}(G)$. Moreover, we could find such a cycle in polynomial time. By the result of Fomin et al. [8], if the length of S is less than $2\delta(H) + k$, then either S can be enlarged in polynomial time, or (a) H is small, that is, $|V(H)| < \text{ad}(H) + k$, yielding that H is extremely dense; or (b) H has a vertex cover of size $\frac{1}{2}\text{ad}(H) - \mathcal{O}(k)$. If S got enlarged, we iterate until we achieve cases (a) or (b). If we are in case (a), the construction of H is completed. In case (b), we need to prune the obtained graph a bit more. More specifically, we can delete $\mathcal{O}(k)$ vertices in the vertex cover and select a subset of the independent set to achieve the property that (i) each of remaining vertices in the vertex cover is adjacent to at least $\text{ad}(H) - \mathcal{O}(k)$ vertices in the selected independent subset, and (ii) every vertex of the selected subset of the independent set sees nearly all vertices of the vertex cover. This means that the obtained induced subgraph is also “dense”, albeit in a different sense. Depending on the case, we use different arguments to establish the routing properties of H .

Routing in H . The case (a), when $|V(H)| < \text{ad}(H) + k$, is easier. In this case, the degrees of almost all vertices are close to $|V(H)|$. Let $S = \{x_1y_1, \dots, x_\ell y_\ell\}$ an arbitrary set of $\mathcal{O}(k)$ pairs of distinct vertices of H forming a linear forest (that is, the union of x_iy_i is a union of disjoint paths). The intuition behind S is that x_i corresponds to the vertex from where the long cycle leaves H and y_i when it enters H again. We show first how to construct a cycle in $H + S$ (that is, the graph obtained from H by turning the pairs of S into edges) containing every pair x_iy_i from S as an edge. This is done by performing constant-length jumps: any two vertices can be connected either by an edge, or through a common neighbor, or through a sequence of two neighbors. Then we extend the obtained cycle to a Hamiltonian cycle in $H + S$ – every vertex of H that is not yet on a cycle can be inserted due to the high degrees of the vertices. The extension of S into a Hamiltonian cycle is shown in Figure 2 (a).



■ **Figure 2** Constructing cycles. The set of pairs S that may be both edges and nonedges of H is shown by red lines and the extension of S into a long cycle is blue. The paths “revolving” around H are green. The vertex cover in c) is denoted by A .

Therefore, if there is a collection of at most k internally vertex disjoint paths going outside from H and returning back, the high density of H allows collecting all of them in a cycle containing all the vertices of H . Together with all the additional vertices these paths visit outside of H we construct a long cycle in G (see Figure 2 (b)). The only condition is that

these paths have to form a linear forest. Thus, if we find a collection of such paths with enough internal vertices, we immediately obtain a long cycle “revolving” around H . The crucial part of the proof is to show that if there is *any* cycle of length at least $\text{ad}(H) + k$ in G , then it can be assumed to have this form.

Let us remark that a similar “rerouting” property was used by Fomin et al. [6] in their above-degeneracy study. Actually, for case (a), we need only a minor adjustment of the arguments from [6]. However, in the “bipartite dense” case (b) the structure of the dense subgraph H is more elaborate and this case requires a new approach. Contrarily to case (a), the long cycle that we construct in $H + S$ is not Hamiltonian but visits all the vertices of the vertex cover (see Figure 2 (c)). In this case, the behavior of paths depends on which part of H they hit. Because of that, while establishing the routing properties, we have to take into account the difference between paths connecting vertices from the vertex cover, independent set, and both. Pushing the “rerouting” intuition through, in this case, turns out to be quite challenging.

Final steps. After finalizing the “rerouting” arguments above, it only remains to design an algorithm that checks whether there exists a collection of paths in G that start and end in H and have at least a certain number of internal vertices in total. We do it by a color-coding-style approach. For case (a), such a subroutine has already been developed in the above-degeneracy case [6]. On the other hand, for the “bipartite dense” case (b) we need to impose an additional restriction on the desired paths, as the length of the final cycle also depends on how the paths’ end-vertices are distributed between the two parts and we have to incorporate these kinds of constraints in our path-finding subroutine.

Finally, to solve LONGEST CYCLE ABOVE MAD, we use the fact that given a graph G , we can find an induced subgraph F with $\text{ad}(F) = \text{mad}(G)$ in polynomial time by the result of Goldberg [13] (see also [12]). Then we find a dense subgraph H of F with the described properties and use H to find a cycle of length at least $\text{mad}(G) + k$.

3 Preliminaries

In this section, we introduce basic notations, and a series of previously-known results that will be helpful to us.

We consider only finite undirected graphs. For a graph G , $V(G)$ and $E(G)$ denote its vertex and edge sets, respectively. Throughout the paper we use $n = |V(G)|$ and $m = |E(G)|$ whenever the considered graph G is clear from the context. For a graph G and a subset $X \subseteq V(G)$ of vertices, we write $G[X]$ to denote the subgraph of G induced by X . We write $G - X$ to denote the graph $G[V(G) \setminus X]$; for a single-element set $X = \{x\}$, we write $G - x$. Similarly, if Y is a set of pairs of distinct vertices, $G - Y = (V(G), E(G) \setminus Y)$. For a set Y of pairs of distinct vertices of G , $G + Y$ denotes the graph $(V(G), E(G) \cup Y)$, that is, the graph obtained by adding the edges in $Y \setminus E(G)$; slightly abusing notation we may denote the pairs of such a set Y in the same way as edges. For a vertex v , we denote by $N_G(v)$ the (*open*) *neighborhood* of v , i.e., the set of vertices that are adjacent to v in G . A set of vertices X is a *vertex cover* of G if for every edge xy of G , $x \in X$ or $y \in X$.

A *path* P in G is a subgraph of G with $V(P) = \{v_0, \dots, v_\ell\}$ and $E(P) = \{v_{i-1}v_i \mid 1 \leq i \leq \ell\}$. We write $v_0v_1 \dots v_\ell$ to denote P ; the vertices v_0 and v_ℓ are *end-vertices* of P , the vertices v_2, \dots, v_ℓ are *internal*, and ℓ is the *length* of P . For a path P with end-vertices s and t , we say that P is an (s, t) -path. Two paths P_1 and P_2 are *internally disjoint* if no internal vertex of one of the paths is a vertex of the other; note that end-vertices may be the same.

For two internally disjoint paths P_1 and P_2 having one common end-vertex, we write P_1P_2 to denote the *concatenation* of P_1 and P_2 . A graph F is a *linear forest* if every connected component of F is a path. Let S be a set of pairs of distinct vertices of G ; they may be either edges or nonedges. We say that S is *potentially cyclable* if $(V(G), S)$ is a linear forest. A *cycle* is a graph C with $V(C) = \{v_1, \dots, v_\ell\}$ for $\ell \geq 3$ and $E(C) = \{v_{i-1}v_i \mid 1 \leq i \leq \ell\}$, where $v_0 = v_\ell$. We may write that $C = v_1 \cdots v_\ell$. A cycle C (a path P , respectively) is *Hamiltonian* if $V(C) = V(G)$ ($V(P) = V(G)$, respectively). A graph G is *Hamiltonian* if it has a Hamiltonian cycle.

A set of vertices S is a *separator* of a connected graph G , if $G - S$ is disconnected. For a positive integer k , G is *k-connected* if $|V(G)| > k$ and for every set S of at most $k - 1$ vertices, $G - S$ is connected. If $S = \{v\}$ is a separator of size one, then v is called a *cut-vertex*. Note, in particular, that a connected graph with at least three vertices is 2-connected if it has no cut-vertex. A *block* of a connected graph with at least two vertices is an inclusion-wise maximal induced subgraph without cut-vertices, that is, either a 2-connected graph or K_2 .

The *degree* of a vertex v in a graph G is $d_G(v) = |N_G(v)|$. The *minimum degree* of G is $\delta(G) = \min\{d_G(v) \mid v \in V(G)\}$. For a nonempty set of vertices X , the *average degree* of X is $\text{ad}_G(X) = \frac{1}{|X|} \sum_{v \in X} d_G(v)$, and the *average degree* of G is $\text{ad}(G) = \text{ad}_G(V(G)) = \frac{2m}{n}$. The *maximum average degree* is $\text{mad}(G) = \max\{\text{ad}(H) \mid H \text{ is induced subgraph of } G\}$.

The following observation about the circumference lower bound $\ell_{EG}(G)$ and the average degree of G is useful for us.

► **Observation 8.** *For every graph G with at least two vertices $\ell_{EG}(G) - 1 \leq \text{ad}(G) < \ell_{EG}(G)$.*

Goldberg [13] proved that, given a graph G , an induced subgraph H of maximum *density*, that is, a subgraph with the maximum value $\frac{|E(H)|}{|V(H)|}$, can be found in polynomial time. This result was improved by Gallo, Grigoriadis, and Tarjan [12]. Note that if H is an induced subgraph of maximum density, then $\text{mad}(G) = \text{ad}(H)$.

► **Proposition 9** ([12]). *An induced subgraph of maximum density of a given graph G can be found in $\mathcal{O}(nm \log(n^2/m))$ time.*

We use the lower bound on the length of a longest (s, t) -path in a 2-connected graph via the average degree obtained by Fan [5].

► **Proposition 10** ([5, Theorem 1]). *Let s and t be two distinct vertices in a 2-connected graph G . Then G has an (s, t) -path of length at least $\text{ad}_G(V(G) \setminus \{s, t\})$.*

Notice that the proof of Proposition 10 in [5] is constructive and a required path can be found in polynomial time.

It is well-known that LONGEST CYCLE, which asks whether a graph has a cycle of length at least k , can be solved in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. The currently best deterministic algorithm is due to Fomin et al. [11].

► **Proposition 11** ([11]). *LONGEST CYCLE can be solved in $4.884^k \cdot n^{\mathcal{O}(1)}$ time.*

The task of LONGEST (s, t) -PATH is, given a graph G with two *terminal* vertices s and t , and a positive integer k , decide whether G has an (s, t) -path with at least k vertices. Fomin et al. [11] proved that this problem is FPT when parameterized by k .

► **Proposition 12** ([11]). *LONGEST (s, t) -PATH can be solved in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time.*

4 Finding a dense subgraph

Here we show that given an instance of LONGEST CYCLE ABOVE MAD, we can in polynomial time either solve the problem or find a dense induced subgraph of the input graph. This part crucially depends on structural and algorithmic results obtained by Fomin et al. in [8]. We derive the following structural corollary for 3-connected graphs from [8, Lemma 20].

► **Corollary 13** (★). *Let G be a 3-connected graph and k be an integer such that $0 < k \leq \frac{1}{24}\delta(G)$. Then there is an algorithm that, given a cycle C of length less than $2\delta(G) + k$, in polynomial time either*

- *returns a longer cycle in G , or*
- *returns a vertex cover of G of size at most $\delta(G) + 2k$, or*
- *reports that C is Hamiltonian.*

Now, by applying exhaustively the classical reduction rules from the proof of Theorem 1 and a new reduction rule that removes sparse 2-connected components, we reach the situation where we can apply Corollary 13. The result of this process is encapsulated in the next lemma.

► **Lemma 14** (★). *There is a polynomial-time algorithm that, given an instance (G, k) of LONGEST CYCLE ABOVE MAD, where $0 < k \leq \frac{1}{80}\text{mad}(G) - 1$, either*

- (i) *finds a cycle of length at least $\text{mad}(G) + k$ in G , or*
- (ii) *finds an induced subgraph H of G with $\text{ad}(H) \geq \text{mad}(G) - 1$ such that $\delta(H) \geq \frac{1}{2}\text{ad}(H)$ and $|V(H)| < \text{ad}(H) + k + 1$, or*
- (iii) *finds an induced subgraph H of G such that there is a partition $\{A, B\}$ of $V(H)$ with the following properties:*
 - *B is an independent set,*
 - *$\frac{1}{2}\text{mad}(G) - 4k \leq |A|$,*
 - *for every $v \in A$, $|N_H(v) \cap B| \geq 2|A|$,*
 - *for every $v \in B$, $d_H(v) \geq |A| - 2k - 2$.*

5 Covering vertices of dense graphs

In this section, we prove that, given a sufficiently dense graph G and a bounded-size set of pairs of distinct vertices S forming a linear forest, we can find a long cycle in $G + S$ containing all edges from S . First, we consider the case where there is a small number of vertices in the graph compared to the average degree. Then, we deal with the case where one part in a bipartition of a dense bipartite graph has bounded size. The proofs of the next two lemmas follow the strategy of using the high density of the graph to connect an arbitrary small subset of vertices in a cycle via constant-length jumps, and then extend this cycle to a long cycle using similar arguments. Recall that for a set S of pairs of distinct vertices of a graph G , we say that S is potentially cyclable if $(V(G), S)$ is a linear forest.

► **Lemma 15** (★). *Let G be a graph and k be an integer such that (i) $0 < k \leq \frac{1}{60}\text{ad}(G)$, (ii) $\delta(G) \geq \frac{1}{2}\text{ad}(G)$, and (iii) $\text{ad}(G) + k > n$. Let also S be a potentially cyclable set of at most k pairs of distinct vertices. Then $G + S$ has a Hamiltonian cycle containing every edge of S .*

Now we consider dense bipartite graphs. Similarly to Lemma 15, we show that for a given set of pairs of vertices forming a linear forest there is a cycle containing all these pairs in the extended graph, and also each vertex of the “high-degree” part of the graph. For an example, see Figure 3.

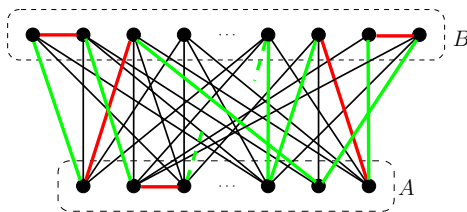


Figure 3 Structure of G and $G' = G + S$. The set of pairs S is shown by red lines and the edges of C that are not in S are green. Note that G' is not required to be bipartite.

► **Lemma 16** (\star). Let G be a bipartite graph, $\{A, B\}$ is a bipartition of $V(G)$ with $p = |A|$, and let k be an integer such that (i) $0 < k \leq \frac{1}{10}p$, (ii) for every $v \in A$, $d_G(v) \geq 2p$, and (iii) for every $v \in B$, $d_G(v) \geq p - k$. Let S be a potentially cyclable set of at most $\frac{9}{4}k$ pairs of distinct vertices. Then $G' = G + S$ has a cycle C containing every edge of S and every vertex of A . Furthermore, C is a longest cycle in G' containing the edges of S and the length of C is $2p - s + t$, where s is the number of edges of S with both end-vertices in A and t is the number of edges in S with both end-vertices in B .

6 Rerouting long cycles to dense subgraphs

In this section, we show that a dense induced subgraph can be used to find a long cycle in a 2-connected graph. Specifically, we show that one can always assume that a long cycle is an extension of a longest cycle in a dense subgraph. To state this more precisely, we need some additional terminology that we introduce next.

Let $T \subseteq V(G)$ for a graph G . A path P is called a T -segment if P has length at least two, the end-vertices of P lie in T , and $v \notin T$ for any internal vertex v of P . A set of internally disjoint paths $\mathcal{P} = \{P_1, \dots, P_r\}$ is a system of T -segments if (i) P_i is a T -segment for every $i \in \{1, \dots, r\}$, and (ii) the union of the paths in \mathcal{P} is a linear forest. Let $A, B \subseteq V(G)$ be disjoint sets of vertices in G . For a pair $\{x, y\}$ of distinct vertices in G , we say that $\{x, y\}$ is an A -pair (B -pair, respectively) if $x, y \in A$ ($x, y \in B$, respectively), and we say that $\{x, y\}$ is an (A, B) -pair if either $x \in A, y \in B$ or, symmetrically, $y \in A, x \in B$. If $\{A, B\}$ is a partition of $T \subseteq V(G)$, then for a T -segment P with end-vertices x and y , P is an A -segment if $\{x, y\}$ is an A -pair, P is a B -segment if $\{x, y\}$ is a B -pair, and P is an (A, B) -segment if $\{x, y\}$ is an $\{A, B\}$ -pair.

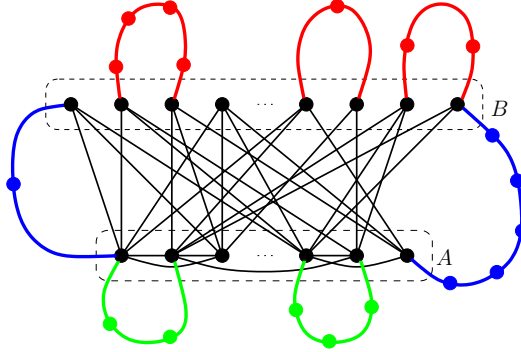
First, we consider the case when there is a dense subgraph H with the property that for every potentially cyclable set S of at most k pairs of distinct vertices, $H + S$ has a Hamiltonian cycle containing every edge of S . We show the following lemma whose proof is almost identical to the proof of Lemma 3 in [6].

► **Lemma 17** (\star). Let G be a 2-connected graph and let k be a positive integer. Suppose that H is an induced subgraph of G such that $|V(H)| \geq 2k$ and for every potentially cyclable set S of at most k pairs of distinct vertices of H , $H + S$ has a Hamiltonian cycle containing every edge of S . Then G has a cycle of length at least $|V(H)| + k$ if and only if one of the following holds:

- (i) There are two distinct vertices $s, t \in V(H)$ such there is an (s, t) -path P in G of length at least $k + 1$ whose internal vertices lie in $V(G) \setminus V(H)$.
- (ii) There is a system of T -segments $\mathcal{P} = \{P_1, \dots, P_r\}$ for $T = V(H)$ such that $r \leq k$ and the total number of vertices on the paths in \mathcal{P} outside T is at least k and at most $2k - 2$.

55:10 Longest Cycle Above Erdős–Gallai Bound

Now we show a related result for dense induced subgraphs of another type. See Figure 4 for an illustration.



■ **Figure 4** Structure of segments in Case (ii) of Lemma 18. The A -segments are shown by green lines, the B -segments are red, and the (A, B) -segments are blue.

► **Lemma 18.** *Let G be a 2-connected graph and let k be a positive integer. Suppose that H is an induced subgraph of G whose set of vertices has a partition $\{A, B\}$ with $|A| \geq \frac{3}{2}k$ and B being an independent set. Suppose also that for every potentially cyclable set S in H of at most k pairs of distinct vertices in H , with s A -pairs and t B -pairs, $H + S$ has a cycle of length at least $2|A| - s + t$. Then G has a cycle of length at least $2|A| + k$ if and only if one of the following holds:*

- (i) *There are two distinct vertices $x, y \in V(H)$ such that H has an (x, y) -path P of length at least $k + 2$ whose internal vertices lie in $V(G) \setminus V(H)$.*
- (ii) *There is a system of T -segments $\mathcal{P} = \{P_1, \dots, P_r\}$ for $T = V(H)$ with s A -segments and t B -segments such that*
 - (a) $r \leq k$,
 - (b) *every A -segment has at least two internal vertices,*
 - (c) *the total number of internal vertices on the paths in \mathcal{P} is at least $k + s - t$ and at most $3k - 2$.*

Proof. Let $T = V(H)$. First, we show that if either (i) or (ii) is fulfilled, then G has a cycle of length at least $|V(H)| + k$.

Suppose that there are distinct $x, y \in T$ and an (x, y) -path P in G with all internal vertices outside T such that the length of P is at least $k + 2$. Let $S = \{xy\}$. We have that $H + S$ has a cycle C containing xy of length at least $2|A| - 1$. We replace the edge xy in C by the path P . Then the length of the obtained cycle C' is at least $2|A| + k$ as required.

Assume that there is a system of T -segments $\mathcal{P} = \{P_1, \dots, P_r\}$ for $T = V(H)$ with s A -segments and t B -segments such that (a)–(b) are fulfilled. Let x_i and y_i be the end-vertices of P_i for $i \in \{1, \dots, r\}$ and define $S = \{x_1y_1, \dots, x_ry_r\}$. Observe that S is a potentially cyclable set for H and $|S| \leq k$. Then $H + S$ has a cycle C of length at least $2|A|$ that contains every edge of S . We construct the cycle C' from C by replacing x_iy_i by the path P_i for every $i \in \{1, \dots, r\}$. Because the total number of internal vertices in the paths of \mathcal{P} is at least $k + s - t$, the length of C' is at least $|V(H)| + k$.

For the opposite direction, assume that G has a cycle C of length at least $2|A| + k$. We consider the following three cases.

Case 1. $V(C) \cap T = \emptyset$. Since G is a 2-connected graph, there are pairwise distinct vertices $x, y \in T$ and $x', y' \in V(C)$, and vertex disjoint (x, x') and (y, y') -paths P_1 and P_2 such that the internal vertices of the paths are outside $T \cup V(C)$. The cycle C has length at least $2|A| + k \geq 3k$. Therefore, C contains an (x', y') -path P with at least $k + 1$ vertices. The concatenation of P_1 , P and P_2 is an (x, y) -path in G of length at least $k + 2$ whose internal vertices are outside T . Hence, (i) is fulfilled.

Case 2. $|V(C) \cap T| = 1$. Let $V(C) \cap T = \{x\}$ for some vertex x . Since G is 2-connected, there is an (y, y') -path P in $G - x$ such that $y' \in V(C)$, $y \in T$, and the internal vertices are outside $T \cup V(C)$. Because the length of C is at least $3k$, C contains an (x', y') -path P' with at least $k + 2$ vertices. The concatenation of P' and P is an (s, t) -path in G of length at least $k + 2$ whose internal vertices are outside T . Hence, (i) holds.

Case 3. $|V(C) \cap T| \geq 2$. Observe that because B is an independent set, H has no cycle of length greater than $2|A|$. Therefore, as $k > 0$ and $|V(C)| \geq 2|A| + k$, $V(C) \setminus T \neq \emptyset$. Let P_1, \dots, P_ℓ be the “outside” segments of C with respect to H , that is, P_1, \dots, P_ℓ are paths on C such that (*) for every $i \in \{1, \dots, \ell\}$, P_i is an (x_i, y_i) -path with at least one internal vertex for some distinct $x_i, y_i \in T$ and the internal vertices of P_i are outside T , and (**) $\bigcup_{i=1}^{\ell} V(P_i) \setminus T = V(C) \setminus T$. If P_i has length at least $k + 2$ for some $i \in \{1, \dots, \ell\}$, then (i) holds. Assume that this is not the case, that is, the length of each P_i is at most $k + 1$. Let $I_A, I_B, I_{AB} \subseteq \{1, \dots, \ell\}$ be the subsets of indices such that P_i is an A -segment for $i \in I_A$, a B -segment for $i \in I_B$, and an (A, B) -segment for $i \in I_{AB}$; note that some of these sets may be empty.

First, we consider I_B . Suppose that the paths P_i for $i \in I_B$ have at least $k - |I_B|$ internal vertices. Consider an inclusion minimal subset of indices $J \subseteq I_B$ such that the paths P_i for $i \in J$ have at least $k - |J|$ internal vertices and let $S = \{x_i y_i \mid i \in J\}$. Observe that the pairs of S compose either a linear forest or a cycle. Suppose that the pairs in S form a cycle. Then every edge of C is outside H , and we have that C is the concatenation of the paths $P_i \in J$. Note that $|J| \geq 2$ in this case. Let $j \in J$. By the choice of J , the total number of internal vertices on the paths P_i for $i \in J \setminus \{j\}$ is at most $k - |J| - 1$. Because the length of P_j is at most $k + 1$, we have that $|V(C)| \leq (k - |J| - 1) + |J| + k = 2k - 1 < 2|A| + k$; a contradiction. Therefore, S forms a linear forest. We obtain that $\mathcal{P} = \{P_i \mid i \in J\}$ is a system of T segments and $|\mathcal{P}| \leq k$. To see that the total number of internal vertices on the paths in \mathcal{P} is at most $2k$, let $j \in J$. Because the total number of internal vertices on the paths P_i for $i \in J \setminus \{j\}$ is at most $k - |J| - 1$ and the length of P_j is at most $k + 1$, the number of internal vertices on the paths in \mathcal{P} is at most $(k - |J| - 1) + k \leq 3k - 2$. We conclude that (ii) is fulfilled.

Assume from now on that the paths P_i for $i \in I_B$ have at most $k - |I_B| - 1$ internal vertices. Then we analyse I_{AB} in a similar way. Let $t = |I_B|$. Suppose that the paths P_i for $i \in I_{AB} \cup I_B$ have at least $k - t$ internal vertices. Consider an inclusion minimal subset of indices $J \subseteq I_{AB}$ such that the paths P_i for $i \in J \cup I_B$ have at least $k - t$ internal vertices and let $S = \{x_i y_i \mid i \in J \cup I_B\}$. Notice that $|S| \leq k$. Again, we have that the pairs of S compose either a linear forest or a cycle. Then we exclude the possibility that S forms a cycle. If we have a cycle, then C is the concatenation of the paths $P_i \in J \cup I_B$. Pick an arbitrary $j \in J$. We have that the total number of internal vertices on the paths P_i for $i \in (J \setminus \{j\}) \cup I_B$ is at most $k - t - 1$. Because the length of P_j is at most $k + 1$, $|V(C)| \leq (k - t - 1) + (|J| + t) + k = 2k + |J| - 1 < 2|A| + k$ and we get a contradiction. Hence, S forms a linear forest and $\mathcal{P} = \{P_i \mid i \in J \cup I_B\}$ is a system of T segments and

$|\mathcal{P}| \leq k$. To upper bound the total number of internal vertices on the paths in \mathcal{P} , let $j \in J$. Because the total number of internal vertices on the paths P_i for $i \in (J \setminus \{j\}) \cup I_B$ is at most $k - t - 1$ and the length of P_j is at most $k + 1$, the number of internal vertices on the paths in \mathcal{P} is at most $2k - t - 1 \leq 3k - 2$. We obtain that (ii) holds.

It remains to consider the case where the paths P_i for $i \in I_{AB} \cup I_B$ have at most $k - t - 1$ internal vertices. For this we analyse I_A . Let $I'_A \subseteq I_A$ be the set of indices $i \in I_A$ such that P_i has at least two internal vertices. Let r be the number of internal vertices on the paths P_i with $i \in I'_A \cup I_B \cup I_{AB}$. Observe that because B is an independent set, $|V(C)| \leq r + t + 2|A| - |I'_A|$. Hence, $r + t - |I'_A| \geq k$. We select an inclusion minimal set of indices $J \subseteq I'_A$ such that the paths P_i for $i \in J \cup I_B \cup I_{AB}$ have at least $k - t + |J|$ internal vertices and let $S = \{x_i y_i \mid i \in J \cup I_B \cup I_{AB}\}$. Let also $s = |J|$. Observe that because P_i has at least two internal vertices for every $i \in I'_A$, $|S| \leq k$. In the same way as above, the pairs of S compose either a linear forest or a cycle, and we show that it should be a linear forest. If the pairs of S form a cycle, then C is the concatenation of the paths $P_i \in J$. Let $j \in J$. By the minimality of J , the total number of internal vertices on the paths P_i for $i \in (J \setminus \{j\}) \cup I_B \cup I_{AB}$ is at most $k + (s - 1) - t - 1$. Because the length of P_j is at most $k + 1$, $|V(C)| \leq (k + (s - 1) - t - 1) + (s + t + |I_{AB}|) + k = 2k + |I_{AB}| + 2s - 2$. Observe that $t + s + |I_{AB}| \leq k$, because if $t + s + |I_{AB}| \geq k + 1$, the total number of the internal vertices on the paths P_i for $i \in (J \setminus \{j\}) \cup I_B \cup I_{AB}$ would be at least $k + s$. Therefore, $|V(C)| \leq 2k + |I_{AB}| + 2s - 2 \leq 4k - 2 < 2|A| + k$; a contradiction. We obtain that S forms a linear forest and $\mathcal{P} = \{P_i \mid i \in J \cup I_B \cup I_{AB}\}$ is a system of T segments, and $|\mathcal{P}| \leq k$. To get the upper bound for the total number of internal vertices on the paths in \mathcal{P} , let $j \in J$. Because the total number of internal vertices on the paths P_i for $i \in (J \setminus \{j\}) \cup I_B$ is at most $k + (s - 1) - t - 1$ and the length of P_j is at most $k + 1$, the number of internal vertices on the paths in \mathcal{P} is at most $2k + s - t - 2 \leq 3k - 2$. We conclude that (ii) is fulfilled. This concludes the analysis of Case 3 and the proof of the lemma. \blacktriangleleft

Fomin et al. [6] proved the following algorithmic result about systems of T -segments.

► **Proposition 19** ([6, Lemma 4]). *Let G be a graph, $T \subseteq V(G)$, and let p and r be positive integers. Then it can be decided in $2^{\mathcal{O}(p)} \cdot n^{\mathcal{O}(1)}$ time whether there is a system of T -segments \mathcal{P} with r paths having p internal vertices in total.*

However, we need an algorithm for constructing a system of T -segments with additional properties described in Lemma 18. For this, we modify the algorithm from Proposition 19.

► **Lemma 20** (\star). *Let G be a graph, $T \subseteq V(G)$, and let $\{A, B\}$ be a partition of T . Let also p and r be positive integers, and suppose that s and t are nonnegative integers with $s + t \leq r$. Then it can be decided in $2^{\mathcal{O}(p)} \cdot n^{\mathcal{O}(1)}$ time whether there is a system of T -segments \mathcal{P} with r paths having p internal vertices in total such that (i) \mathcal{P} contains s A -segments, (ii) t B -segments, and (iii) every A -segment has at least two internal vertices.*

7 Proof of the main result

Now we have all ingredients to prove our main result. We restate it here for the reader's convenience.

► **Theorem 2.** *LONGEST CYCLE ABOVE MAD can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ on 2-connected graphs.*

Proof. Let (G, k) be an instance of LONGEST CYCLE ABOVE MAD, where G is a 2-connected graph. We use the algorithm from Proposition 9 and compute $\text{mad}(G)$ in polynomial time. If $k = 0$, the problem is trivial, because a cycle of length at least $\text{mad}(G)$ exists by Theorem 1. Hence, we can assume that $k \geq 1$. If $k > \frac{1}{88}\text{mad}(G) - 1$, we use Proposition 11 and solve the problem in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. From now, we assume that $0 < k \leq \frac{1}{88}\text{mad}(G) - 1$. In particular, $k \leq \frac{1}{80}\text{mad}(G) - 1$. We apply Lemma 14, and in polynomial time either

- (i) find a cycle of length at least $\text{mad}(G) + k$ in G , or
- (ii) find an induced subgraph H of G with $\text{ad}(H) \geq \text{mad}(G) - 1$ such that $\delta(H) \geq \frac{1}{2}\text{ad}(H)$ and $|V(H)| < \text{ad}(H) + k + 1$, or
- (iii) find an induced subgraph H of G such that there is a partition $\{A, B\}$ of $V(H)$ with the following properties:
 - B is an independent set,
 - $\frac{1}{2}\text{mad}(G) - 4k \leq |A|$,
 - for every $v \in A$, $|N_H(v) \cap B| \geq 2|A|$,
 - for every $v \in B$, $d_H(v) \geq |A| - 2k - 2$.

If the algorithm finds a cycle of length at least $\text{mad}(G) + k$, then we return it and stop. In Cases (ii) and (iii), we get a dense induced subgraph H that can be used to find a solution.

Case (ii). The algorithm from Lemma 14 returns an induced subgraph H of G with $\text{ad}(H) \geq \text{mad}(G) - 1$ such that $\delta(H) \geq \frac{1}{2}\text{ad}(H)$ and $|V(H)| < \text{ad}(H) + k + 1$. Let $k' = \lceil \text{mad}(G) \rceil + k - |V(H)|$. We have that G has a cycle of length at least $\text{mad}(G) + k$ if and only if G has a cycle of length at least $|V(H)| + k'$. Note that $k' \leq k + 1 \leq \frac{1}{88}\text{mad}(G) \leq \frac{1}{60}\text{ad}(H)$. By Lemma 15, for every potentially cyclable set S of at most $k + 1$ pairs of distinct vertices of H , $H + S$ has a Hamiltonian cycle containing every edge of S .

Suppose that $k' \leq 0$. Observe that H has a Hamiltonian cycle as we can use Lemma 15 for $S = \{e\}$, where e is an arbitrary edge $e \in E(H)$. Then we conclude that H has a cycle of length at least $\text{mad}(G) + k$ and stop. Assume that $k' > 0$. Note that $|V(H)| \geq \text{ad}(H) \geq 2k'$. Then by Lemma 17, G has a cycle of length at least $|V(H)| + k'$ if and only if one of the following holds:

- (a) There are two distinct vertices $s, t \in V(H)$ such that H has an (s, t) -path P of length at least $k' + 1$ whose internal vertices lie in $V(G) \setminus V(H)$.
- (b) There is a system of T -segments $\mathcal{P} = \{P_1, \dots, P_r\}$ for $T = V(H)$ such that $r \leq k'$ and the total number of vertices on the paths in \mathcal{P} outside T is at least k' and at most $2k' - 2$.

First, we check if (a) can be satisfied. For this, we consider all pairs of distinct vertices s and t of H . For every pair, we construct $G' = G[(V(G) \setminus V(H)) \cup \{s, t\}]$ and use Proposition 12 to find an (s, t) -path of length at least $k' + 1$ in G' in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. If we find such a path for some pair, we report the existence of a cycle of length at least $\text{mad}(G) + k$ and stop. Otherwise, we verify (b) using Proposition 19. We use the algorithm from Proposition 19 for $r \in \{1, \dots, k'\}$ and for $p \in \{k', \dots, 2k' - 2\}$. If we find a required system of T -segments, then we return that G has a cycle of length at least $\text{mad}(G) + k$ and stop. If we fail to find such a system for every r and p , we conclude that G has no cycle of length at least $\text{mad}(G) + k$. Note that this can be done in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. This concludes Case (ii).

Case (iii). The algorithm from Lemma 14 returns an induced subgraph H of G such that there is a partition $\{A, B\}$ of $V(H)$ with the properties:

- B is an independent set,
- $\frac{1}{2}\text{mad}(G) - 4k \leq |A|$,
- for every $v \in A$, $|N_H(v) \cap B| \geq 2|A|$,
- for every $v \in B$, $d_H(v) \geq |A| - 2k - 2$.

Let $k' = \lceil \text{mad}(G) \rceil + k - 2|A|$. Observe that G has a cycle of length at least $\text{mad}(G) + k$ if and only if G has a cycle of length at least $2|A| + k'$. We have that $2|A| \geq \lceil \text{mad}(G) \rceil - 8k$ and, therefore, $k' \leq 9k$.

Note that $|A| \geq \frac{1}{2}\text{mad}(G) - 4k \geq 40k$, since $k \leq \frac{1}{88}\text{mad}(G) - 1$. Also, we have that for every $v \in B$, $d_H(v) \geq |A| - 4k$. Therefore, by Lemma 16, for every potentially cyclable set S of at most $9k$ pairs of distinct vertices, $G' = G + S$ has a cycle C containing every edge of S and the length of C is $2|A| - s + t$, where s is the number of edges of S with both end-vertices in A and t is the number of edges in S with both end-vertices in B .

Suppose that $k' \leq 0$. Then we observe that H has a cycle of length $2|A|$ because we can set $S = \{xy\}$, where $xy \in E(H)$ with $x \in A$ and $y \in B$. Then H has a cycle of length at least $2|A| + k'$ and we conclude that G has a cycle of length at least $\text{mad}(G) + k$. Assume that $k' > 0$. Since $|A| \geq 40k \geq \frac{3}{2}k'$, we can apply Lemma 18. We obtain that G has a cycle of length at least $2|A| + k'$ if and only if one of the following holds:

- (a) There are two distinct vertices $x, y \in V(H)$ such that H has an (x, y) -path P of length at least $k' + 2$ whose internal vertices are in $V(G) \setminus V(H)$.
- (b) There is a system of T -segments $\mathcal{P} = \{P_1, \dots, P_r\}$ for $T = V(H)$ with s A -segments and t B -segments such that
 - $r \leq k' \leq 9k$,
 - every A -segment has at least two internal vertices,
 - the total number of internal vertices on the paths in \mathcal{P} is at least $k' + s - t$ and at most $3k' - 2 \leq 27k - 2$.

To verify (a), we use the same approach as in Case (ii), that is, we consider all pairs of distinct vertices x and y of H . For every pair, we construct $G' = G[(V(G) \setminus V(H)) \cup \{x, y\}]$ and use Proposition 12 to find an (x, y) -path of length at least $k' + 2$ in G in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time. If we find such a path for some pair, we report the existence of a cycle of length at least $\text{mad}(G) + k$ and stop. Otherwise, we verify (b) using Lemma 20. We use the algorithm from this lemma for $r \in \{1, \dots, k'\}$, $s \in \{0, \dots, k'\}$ and $t \in \{0, \dots, k'\}$ such that $s + t \leq r$, and for $p \in \{k' + s - t, 3k' - 2\}$. If we find a system of T -segments $\mathcal{P} = \{P_1, \dots, P_r\}$ for $T = V(H)$ with s A -segments and t B -segments with the required properties, then we conclude that G has a cycle of length at least $2|A| + k'$ and stop. If such a system does not exist for every choice of r , s , t , and p , we have that G has no cycle of length at least $\text{mad}(G) + k$. By Lemma 20, this can be done in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ time, because $k' \leq 9k$. This concludes Case (iii).

Because the algorithm from Lemma 14 is polynomial and the other subroutines used in our algorithm for LONGEST CYCLE ABOVE MAD run in $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, the overall running time is $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ and this concludes the proof.

Let us remark that since the algorithms for paths in Propositions 19 and 12 and Lemma 20 are, in fact, constructive, and the same holds for the algorithms for cycles in Lemmas 17 and 18 and Proposition 11, our algorithm is not only able to solve the decision problem, but also can find a cycle of length at least $\text{mad}(G) + k$ if it exists. ◀

References

- 1 Ivona Bezáková, Radu Curticapean, Holger Dell, and Fedor V. Fomin. Finding detours is fixed-parameter tractable. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPICs*, pages 54:1–54:14. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017.
- 2 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

- 3 Gabriel A. Dirac. Some theorems on abstract graphs. *Proc. London Math. Soc. (3)*, 2:69–81, 1952.
- 4 Paul Erdős and Tibor Gallai. On maximal paths and circuits of graphs. *Acta Math. Acad. Sci. Hungar.*, 10:337–356 (unbound insert), 1959.
- 5 Genghua Fan. Long cycles and the codiameter of a graph, I. *J. Comb. Theory, Ser. B*, 49(2):151–180, 1990. doi:10.1016/0095-8956(90)90024-T.
- 6 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Going far from degeneracy. *SIAM J. Discret. Math.*, 34(3):1587–1601, 2020. doi:10.1137/19M1290577.
- 7 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Parameterization above a multiplicative guarantee. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151 of *LIPICs*, pages 39:1–39:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ITCS.2020.39.
- 8 Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. Algorithmic extensions of Dirac’s theorem. *CoRR*, abs/2011.03619, 2020. arXiv:2011.03619.
- 9 Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. Algorithmic extensions of Dirac’s theorem. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, (SODA 2022)*, pages 931–950. SIAM, 2022. doi:10.1137/1.9781611977073.20.
- 10 Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. Longest cycle above Erdős-Gallai bound. *CoRR*, abs/2202.03061, 2022. arXiv:2202.03061.
- 11 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Long directed (s, t) -path: FPT algorithm. *Inf. Process. Lett.*, 140:8–12, 2018. doi:10.1016/j.ipl.2018.04.018.
- 12 Giorgio Gallo, Michael D. Grigoriadis, and Robert Endre Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, 1989. doi:10.1137/0218003.
- 13 Andrew V. Goldberg. Finding a maximum density subgraph. Technical Report CSD-84-171, University of California, 1984.
- 14 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 15 Bart M. P. Jansen, László Kozma, and Jesper Nederlof. Hamiltonicity below Dirac’s condition. In *Proceedings of the 45th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 11789 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2019.

Improved Polynomial-Time Approximations for Clustering with Minimum Sum of Radii or Diameters

Zachary Friggstad ✉

Department of Computing Science, University of Alberta, Edmonton, Canada

Mahya Jamshidian ✉

Department of Computing Science, University of Alberta, Edmonton, Canada

Abstract

We give an improved approximation algorithm for two related clustering problems. In the Minimum Sum of Radii clustering problem (MSR), we are to select k balls in a metric space to cover all points while minimizing the sum of the radii of these balls. In the Minimum Sum of Diameters clustering problem (MSD), we are to simply partition the points of a metric space into k parts while minimizing the sum of the diameters of these parts. We present a 3.389-approximation for MSR and a 6.546-approximation for MSD, improving over their respective 3.504 and 7.008 approximations developed by Charikar and Panigrahy (2001). In particular, our guarantee for MSD is better than twice our guarantee for MSR.

Our approach refines a so-called bipoint rounding procedure of Charikar and Panigrahy's algorithm by considering centering balls at some points that were not necessarily centers in the bipoint solution. This added versatility enables the analysis of our improved approximation guarantees. We also provide an alternative approach to finding the bipoint solution using a straightforward LP rounding procedure rather than a primal-dual algorithm.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Approximation Algorithms, Clustering, Linear Programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.56

Funding *Zachary Friggstad*: Supported by an NSERC Discovery Grant and Discovery Accelerator Supplement.

Acknowledgements The first author would like to thank Mohammad R. Salavatipour for preliminary discussions on this problem.

1 Introduction

Clustering, as one of the fundamental problems in information technology, has been studied in computing science and several other fields to a great extent. Different methods of clustering have been used significantly in data mining, bioinformatics, pattern recognition, computer vision, etc. The goal of clustering is to partition a set of data points into partitions, called clusters. Many of clustering problems involve finding k cluster centers and a mapping σ from data points to the centers to minimize some objective function. One of the most studied such objective functions is k -CENTER which minimizes the maximum diameter (or radius) [9, 18]. Another examples is the k -MEDIAN problem which aims to minimize sum of distances from data points to their centers, as extensively studied in [5, 6, 19, 20, 21].

In this paper, we focus on a different objective function for clustering that is more center-focused in that the cost of a cluster is the radius of the ball used to cover that cluster. Specifically, we study the following problem.



© Zachary Friggstad and Mahya Jamshidian;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, 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

► **Definition 1.** In the MINIMUM SUM OF RADII problem (MSR), we are given a set X of n points in a metric space with distances d and a positive integer k . We are to select centers $C \subseteq X$, $|C| \leq k$ and assign each $i \in C$ a radius r_i so that each $j \in X$ lies within distance r_i of at least one $i \in C$ (i.e. $d(j, i) \leq r_i$). The goal is to minimize the total radii, i.e. $\sum_{i \in C} r_i$.

We also consider the related problem to minimize sum of diameters of the clusters chosen. Note this variant is simply about partitioning the point set, there are no centers involved.

► **Definition 2.** In the MINIMUMS SUM OF DIAMETERS problem (MSD), the input is the same as in MSR and our goal is to partition the points into k clusters X_1, X_2, \dots, X_k to minimize $\sum_{i=1}^k \max_{j, j' \in X_i} d(j, j')$, the sum of the diameters of the clusters.

It is easy to see that an α -approximation algorithm for MSR yields a 2α -approximation algorithm for MSD. That is, if OPT_R denotes the optimum MSR solution cost and OPT_D an optimum MSD solution cost, we have $OPT_R \leq OPT_D$ because in the optimum MSD solution we could pick any point from each cluster to act as its center (with radius equal to the diameter of the cluster). So if we have an MSR solution with cost at most $\alpha \cdot OPT_R$, then if we define clusters X_i by sending each point to some center whose ball covers that point, the diameter of cluster i would be $\leq 2 \cdot r_i$ so the sum of diameters would then be at most $2\alpha \cdot OPT_R \leq 2\alpha \cdot OPT_D$.

1.1 Our Contributions

Prior to this work, Charikar and Panigrahy presented a 3.504-approximation for MSR [7]. Since an α -approximation for MSR yields a 2α -approximation for MSD, this also yields a 7.008-approximation for MSD. These were the best polynomial-time approximations for these problems in general metrics.

In this paper, we first present an improved polynomial-time approximation algorithm for MSR. Specifically, we prove the following.

► **Theorem 3.** *There is a polynomial-time 3.389-approximation for MSR.*

We obtain this primarily by refining a so-called bipoint rounding step from [7]. That is, our improvement for MSR mainly focuses in the last phase of the algorithm in [7] which combines two subsets of balls that, together, open an *average* of k centers and whose average cost is low. Their algorithm focuses on selecting k of the centers from these two subsets. We expand the set of possible centers to choose and consider some that may not be centers in the averaging of the two subsets.

While not our main result, we also present an alternative way to obtain these two subsets of balls in that we consider a straightforward rounding of a linear programming (LP) relaxation, the Lagrangian relaxation of the problem obtained by relaxing the constraint that at most k centers are chosen, rather than a primal-dual technique as in [7]. Our rounding algorithm is incredibly simple and we employ fairly generic arguments to convert it to a bipoint solution for a single Lagrangian multiplier λ . This may be of independent interest as it should be easy to adapt to other settings where one wants to get a bipoint solution where both points are obtained from a common Lagrangian value λ , as long as the LMP approximation is from direct LP rounding. We emphasize this is only an alternative approach: we could work directly with their primal-dual approach.

Our second result is an improved MSD approximation that does not just use our MSR approximation as a black box.

► **Theorem 4.** *There is a polynomial-time 6.546-approximation for MSD.*

In particular, notice the guarantee is better than twice our approximation guarantee for MSR. This is obtained through a variation of our new ideas behind our MSR approximation.

We emphasize this is the first improvement to the approximation guarantee from polynomial-time algorithms for these problems in over 20 years.

1.2 Related Work

Gibson et al. show MSR is **NP**-hard even in metrics with constant doubling dimension or shortest-path metrics of edge-weighted planar graphs [10]. In polynomial time, the best approximation algorithm is the stated 3.504 approximation by Charikar and Panigrahy [7]. Interestingly, [10] show that MSR can be solved exactly in $n^{O(\log n \cdot \log \Gamma)}$ where Γ is the *aspect ratio* of the metric (maximum distance divided by minimum nonzero distance). By standard techniques, this yields a quasi-PTAS for MSR: i.e. a $(1 + \epsilon)$ -approximation with running time $n^{O(\log 1/\epsilon + \log^2 n)}$. A major open problem is to design a PTAS for MSR, or perhaps to demonstrate there is no PTAS for MSR under some strong lower bound (eg. the exponential-time hypothesis). For now, it is of interest to get improved constant-factor approximations for MSR. By way of analogy, the *unsplittable flow* problem was known to admit a quasi-PTAS [1, 2] yet improved constant-factor approximations were subsequently produced [15, 13, 14], that is until a PTAS was finally found by Grandoni et al. [12].

On the other hand MSD is hard to approximate: Doddi et al. show that unless $\mathbf{P} = \mathbf{NP}$, there is no $(2 - \epsilon)$ -approximation for MSD for any $\epsilon > 0$ even if the metric is the shortest path metric of an unweighted graph [8]. Prior to our work, the best approximation for MSD is simply twice the best polynomial-time approximation for MSR, i.e. $2 \cdot 3.504 = 7.008$ using the approximation for MSR from [7].

MSR and MSD have been studied in special cases as well. In constant-dimensional Euclidean metrics, MSR can be solved exactly in polynomial time [11]. This is particularly interesting in light of the fact that MSR is hard in doubling metrics. For MSD in constant-dimensional Euclidean metrics, if k is also regarded as a constant then MSD can be solved exactly [4]. In general metrics with $k = 2$, MSD can be solved exactly by observing that if we are given the diameters of the two clusters, we can use 2SAT to determine if we can place the points in these clusters while respecting the diameters [16]. However, MSD is **NP**-hard for even $k = 3$ as it captures the problem of determining if an unweighted graph can be partitioned into 3 cliques. Finally, if one does not allow balls with radius 0 in the solution, MSR can be solved in polynomial time in shortest path metrics of unweighted graphs [3, 17].

1.3 Organization

Our MSR approximation is given in Section 2. Our algorithm follows the same general structure as the algorithm in [7] so we defer the details behind one significant step (obtaining the “bipoint” solution) and focus on our new ideas. Our MSD approximation is then given in Section 3. Finally, our new approach to obtaining a clean bipoint solution is summarized in Section 4. For the sake of space, many proofs in Section 4 are deferred to the full version. Brief concluding remarks are given in Section 5.

2 Minimum Sum of Radii

2.1 Preliminaries

Throughout, $n = |X|$. We assume $d(i, i') > 0$ for distinct $i, i' \in X$ (i.e. there are no collocated points), clearly this is without loss of generality since we could restrict X to contain only one point from each group of collocated points. A **ball** in X is a set of the form $B(i, r) = \{j \in X : d(i, j) \leq r\}$ for some point $i \in X$ and radius $r \geq 0$. We sometimes also call a pair (i, r) a ball with the understanding it is referring to the set $B(i, r)$. One can view a solution to MSR as being a collection of balls.

In some places in our algorithm, we need to guess balls from the optimal solution or use LP variables corresponding to balls that may appear in the optimal solution: in these steps we only need to consider balls $B(i, r)$ where $r = d(i, j)$ for some $j \in X$ because it is clear that an optimal MSR solution will set each radius as to be the furthest point that is covered by that ball. So there are only $O(|X|^2)$ different balls to consider. We view a solution as a collection \mathcal{B} of pairs $(i, r), i \in X, r \geq 0$ describing the centers and radii of balls. For such a subset, we let $\text{cost}(\mathcal{B}) = \sum_{(i,r) \in \mathcal{B}} r$ be the total radii of these balls.

Fix some small constant $\epsilon > 0$ such that $1/\epsilon$ is an integer. Smaller ϵ lead to better guarantees with increased (but still polynomial) running times. Since the bound in the statement of Theorem 3 is just a rounded up version of the actual approximation guarantee, we will ultimately pick ϵ to be small enough to hide it in the approximation guarantee, which is why it does not appear in the statement. We assume $k > 1/\epsilon$, otherwise we can simply use brute force to find the optimum solution in $n^{O(1/\epsilon)}$ time.

Our algorithm for MSR is summarized in Algorithm 1 at the end of this section, though it makes reference to a fundamental subroutine to find our “bipoint” solution that we describe in Section 4. By bipoint, we simply mean two subsets of balls $\mathcal{B}_1, \mathcal{B}_2$ with $|\mathcal{B}_1| \geq k \geq |\mathcal{B}_2|$ so, in a sense to be described later, some averaging of these sets looks like a feasible fractional solution using exactly k balls.

2.2 Step 1: Guessing the Largest Balls

Let \mathcal{B}^* denote some fixed optimum solution with $\text{OPT} := \text{cost}(\mathcal{B}^*)$. Among all optimal solutions, we assume \mathcal{B}^* has the fewest balls. Thus, for distinct $(i, r), (i', r') \in \mathcal{B}^*$ we have that $i' \notin B(i, r)$ since, otherwise, $\mathcal{B}^* - \{(i, r), (i', r')\} \cup \{(i, r + r')\}$ is another optimal solution with even fewer balls.

Similar to [7], we guess the $1/\epsilon$ largest balls in \mathcal{B}^* by trying each subset \mathcal{B}' of $1/\epsilon$ balls and proceeding with the algorithm we describe in the rest of this paper. Let R_m be the minimum radius of a ball in \mathcal{B}' and note $R_m \leq \epsilon \cdot \text{OPT}$. We also let $k' := k - 1/\epsilon$, which is an upper bound on the number of balls in $\mathcal{B}^* - \mathcal{B}'$.

We now restrict ourselves to the instance with points $X' := X - \cup_{(i,r) \in \mathcal{B}'} B(i, r)$ to be covered. Since no center of a ball in \mathcal{B}^* is contained within another ball from \mathcal{B}^* , the remaining balls in $\mathcal{B}^* - \mathcal{B}'$ are also centered in X' . We will let $\text{OPT}' = \text{OPT} - \cup_{(i,r) \in \mathcal{B}'} r$ denote the optimal solution value to this restricted instance. The solution $\mathcal{B}^* - \mathcal{B}'$ for this instance satisfies $r \leq R_m \leq \epsilon \cdot \text{OPT}$ for any $(i, r) \in \mathcal{B}^* - \mathcal{B}'$. We also assume $|X'| > k'$, otherwise we just open zero-radius ball at each point in X' .

Before proceeding to the main part of the algorithm, we perform a “precheck” for this guess as follows: run a standard 2-approximation for the k' -MEDIAN instance on the metric restricted to X' (eg. [18]). If the solution returned has radius $> 2 \cdot R_m$, then we reject this guess \mathcal{B}' . This is valid because we know for a correct guess that the remaining points can each be covered using at most k' balls each with with radius at most R_m . From now on, we let \mathcal{A} denote the k' centers returned by this approximation: so each $j \in X'$ lies in at least one ball of the form $B(i, 2 \cdot R_m)$ for some $i \in \mathcal{A}$.

Summary. After guessing \mathcal{B}' we have restricted ourselves to an MSR instance with points X' , a bound k' on the number of balls to choose (where $k' < |X'|$), and a bound R_m . For a correct guess of \mathcal{B}' , there is an optimal solution that uses balls with radius at most R_m . Furthermore, \mathcal{A} is a set of k' centers such that every $j \in X'$ has $d(j, \mathcal{A}) \leq 2 \cdot R_m$ balls.

When analyzing the rest of the algorithm, will assume that \mathcal{B}' is guessed correctly, i.e. $\mathcal{B}' \subseteq \mathcal{B}^*$ and all $(i, r) \in \mathcal{B}^* - \mathcal{B}'$ have $r \leq R_m$. Our final solution will be the minimum-cost solution found over all guesses \mathcal{B}' that were not rejected, so it will be at most the cost of the solution found when \mathcal{B}' was guessed correctly.

2.3 Step 2: Getting a Bipoint Solution

The output from this step is similar to [7], except we obtain it with a different algorithm. We note that their approach would suffice for our purposes, our reasons for considering this different approach are described after the statement of Theorem 6 below. Some details are deferred to Section 4 and some to the full version of this paper, here we explain what is required to understand our ideas that lead to the improved approximation guarantee.

For a value $\lambda \geq 0$, $\mathbf{LP}(\lambda)$ is the linear program that results by considering the Lagrangian relaxation of MSR. That is, the LP has variables for each possible ball we may add except instead of restricting the number of balls to be at most k' , we simply pay λ for each ball.

Note. Terms of the LP that consider pairs (i, r) corresponding to balls with $i \in X'$ and r of the form $d(i, j)$ for some $j \in X'$ but only for those where $r \leq R_m$. Thus, the LP has $O(|X|^2)$ variables.

$$\begin{aligned} \min \quad & \sum_{(i,r)} (r + \lambda) \cdot x_{i,r} \\ \text{s.t.} \quad & \sum_{(i,r): j \in B(i,r)} x_{i,r} \geq 1 \quad \forall j \in X' \\ & x \geq 0 \end{aligned} \tag{LP(\lambda)}$$

The following is standard and follows by considering the natural integer solution corresponding to the balls in $\mathcal{B}^* - \mathcal{B}'$.

► **Lemma 5.** *For any $\lambda \geq 0$, let $OPT_{\mathbf{LP}(\lambda)}$ denote the optimum value of $\mathbf{LP}(\lambda)$. Then $OPT_{LP(\lambda)} - \lambda \cdot k' \leq OPT'$.*

We summarize the main properties of a bipoint solution that is required by our algorithm. The proof of the following is the subject of Section 4.

► **Theorem 6.** *There is a polynomial-time algorithm that will compute a single value $\lambda \geq 0$ and two sets of balls $\mathcal{B}_1, \mathcal{B}_2$ having respective sizes k_1, k_2 where $k_1 \geq k' \geq k_2$. Furthermore, for every $(i, r) \in \mathcal{B}_1$, there is some $(i', r') \in \mathcal{B}_2$ such that $B(i, r) \cap B(i', r') \neq \emptyset$. Finally, for both $\ell = 1$ and $\ell = 2$ we have the following properties:*

- for each $(i, r) \in \mathcal{B}_\ell$, we have $r \leq 3 \cdot R_m$,
- tripling the radii of each $(i, r) \in \mathcal{B}_\ell$ will cover X' , i.e. for each $j \in X'$ there is some $(i, r) \in \mathcal{B}_\ell$ such that $j \in B(i, 3 \cdot r)$, and
- $cost(\mathcal{B}_\ell) + \lambda \cdot k_\ell \leq OPT_{\mathbf{LP}(\lambda)}$

Again, we note that essentially the same result is found in [7], except it is slightly more technical to state since the two sets \mathcal{B}_1 and \mathcal{B}_2 are obtained through a LMP algorithm applied to different (but very close) values λ_1 and λ_2 which leads to an additional ϵ -loss in the approximation guarantee. Qualitatively speaking, the theorem statement itself is not new and the reader who is not interested in seeing a new technique can skip reading its proof.

As an easy warmup, notice that if $k_1 = k'$ then by Lemma 5 we have

$$cost(\mathcal{B}_1) \leq OPT_{\mathbf{LP}(\lambda)} - \lambda \cdot k' \leq OPT'.$$

In this case, tripling the radii of all balls in \mathcal{B}_1 covers all of X' with cost at most $3 \cdot OPT'$. Together with \mathcal{B}' , this is a feasible MSR solution with cost at most $3 \cdot OPT$. A similar approximation follows if $k_2 = k$. However, we do not distinguish these case in our full analysis below.

2.4 Step 3: Combining Bipoint Solutions

Let $\lambda, \mathcal{B}_1, \mathcal{B}_2$ be the *bipoint solution* from Theorem 6. For brevity, let $C_1 = \text{cost}(\mathcal{B}_1)$ and $C_2 = \text{cost}(\mathcal{B}_2)$. Since $k_1 \geq k' \geq k_2$, there are values $a, b \geq 0$ with $a+b = 1$ and $a \cdot k_1 + b \cdot k_2 = k'$. We fix these values throughout this section.

The following shows the *average* cost C_1 and C_2 is bounded by OPT' , the first inequality is by the last property listed in Theorem 6 and the second by Lemma 5.

$$a \cdot C_1 + b \cdot C_2 \leq a \cdot (OPT_{\text{LP}(\lambda)} - \lambda \cdot k_1) + b \cdot (OPT_{\text{LP}(\lambda)} - \lambda \cdot k_2) = OPT_{\text{LP}(\lambda)} - \lambda \cdot k \leq OPT' \quad (1)$$

The rest of our algorithm and analysis considers how to convert the two solutions $\mathcal{B}_1, \mathcal{B}_2$ to produce a feasible solution whose value is within a constant-factor of this averaging of C_1, C_2 . First, note tripling the radii in all balls in \mathcal{B}_2 will produce a feasible solution as $k_2 \leq k'$, but it may be too expensive. So we will consider two different solutions and take the better of the two. The first solution is what we just described: formally it is $\{(i, 3r) : (i, r) \in \mathcal{B}_2\}$, which is a feasible solution with cost $3 \cdot C_2$.

Constructing the second solution is our main deviation from the work in [7]. Intuitively, we want to cover all points by using balls $(i, 3 \cdot r)$ for $(i, r) \in \mathcal{B}_1$. The cheaper of this and the first solution can easily be shown to have cost at most $3 \cdot OPT'$. The problem is that this could open more than k' centers (if $k_1 > k'$). As in [7], we consolidate some of these balls into a single group based on their common intersection with some $(i', r') \in \mathcal{B}_2$. We will select some groups and merge their balls into a single ball so the number of balls is at most k' . Our improved approximation is enabled by considering different ways to cover balls in a group using a single ball, [7] only considered one possible way to cover a group with a single ball.

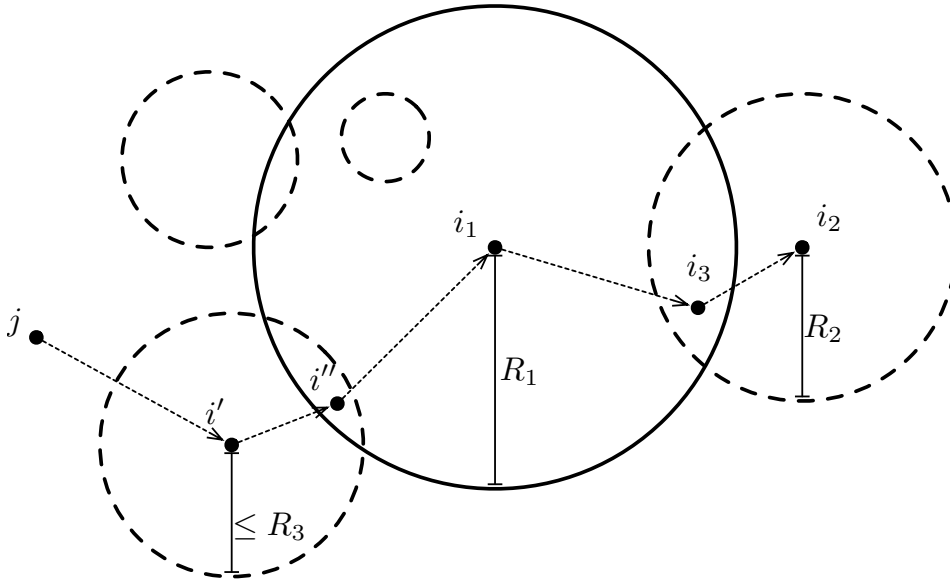
We now form groups. For each $(i, r) \in \mathcal{B}_2$, we create a group $G_{i,r} \subseteq \mathcal{B}_1$ as follows: for each $(i', r') \in \mathcal{B}_1$, consider any single $(i, r) \in \mathcal{B}_2$ such that $B(i, r) \cap B(i', r') \neq \emptyset$ and add (i', r') to $G_{i,r}$. If multiple $(i, r) \in \mathcal{B}_2$ satisfy this criteria, pick one arbitrarily. Let $\mathcal{G} = \{G_{i,r} : (i, r) \in \mathcal{B}_2 \text{ s.t. } G_{i,r} \neq \emptyset\}$ be the collection of all nonempty groups formed this way, note \mathcal{G} is a partitioning of \mathcal{B}_1 .

Covering a group with a single ball

From here, the approach in [7] would describe how to merge the balls in a group $G_{i,r} \in \mathcal{G}$ simply by centering a new ball at i , and making its radius sufficiently large to cover all points covered by the tripled balls $B(i', 3r')$ for $(i', r') \in G_{i,r}$. We consider choosing a different center when we consolidate the \mathcal{B}_1 balls in a group. In fact, it suffices to simply pick the minimum-radius ball that covers the union of the tripled balls in a group. This ball can be centered at any point in X' .

To analyze this, we describe a few candidate balls and argue that the cheapest of these has cost at most $\frac{11}{8} \cdot r + 3 \cdot \text{cost}(G_{i,r})$ for each $G_{i,r} \in \mathcal{G}$. The exact choice of ball we use for the analysis depends on the composition of the group, namely the total and maximum radii of balls in $G_{i,r}$ versus the radius r itself. In [7], the ball they select has cost at most $r + 4 \cdot \text{cost}(G_{i,r})$. While our analysis has a higher dependence on r , when considered as an alternative solution to the one that just triples all balls in \mathcal{B}_2 we end up with a better overall solution.

For now, fix a single group $G_{i,r} \in \mathcal{G}$. Let R_1 denote r , R_2 be the maximum radius of a ball in $G_{i,r}$ and R_3 be the maximum radius among all other balls in $G_{i,r}$ apart from the one defining R_2 . If $G_{i,r}$ has only one ball, then let $R_3 = 0$. That is, $0 \leq R_3 \leq R_2$ but it could be that $R_3 = R_2$, i.e. there could be more than one ball from $G_{i,r}$ with maximum radius. We also let i_1 denote i , i_2 be the center of any particular ball with maximum radius in $G_{i,r}$, and i_3 be any single point in $B(i_1, R_1) \cap B(i_2, R_2)$. There is at least one since each ball in $G_{i,r}$ intersects $B(i, r)$ by construction of the groups.



■ **Figure 1** A depiction of a group G_{i_1, R_1} . The solid ball is $B(i_1, R_1)$ and the dashed balls are those in G_{i_1, R_1} . Point j is covered by tripling the ball centered at i' . The dashed path depicts the way we bound $d(j, i_2)$ in the second part of the case **Centering at i_2** .

Next we describe the radius of a ball that would be required if we centered it at one of i_1, i_2 or i_3 . Consider any $j \in Y_{i, r}$ with, say, $j \in B(i', 3r')$ for some $(i', r') \in G_{i, r}$. Let i'' be any point in $B(i_1, r) \cap B(i', r')$. We bound the distance of j from each of i_1, i_2 and i_3 to see what radius would suffice for each of these three possible centers. Figure 1 depicts this group and one case of the analysis below.

- **Centering at i_1 .** Simply put,

$$\begin{aligned} d(j, i_1) &\leq d(j, i') + d(i', i'') + d(i'', i_1) \\ &\leq 3 \cdot R_2 + R_2 + R_1 \\ &= R_1 + 4 \cdot R_2. \end{aligned}$$

So radius $C^{(1)} := R_1 + 4 \cdot R_2$ suffices if we choose i_1 as the center.

- **Centering at i_2 .** If $(i', r') = (i_2, R_2)$ then $d(j, i_2) \leq 3 \cdot R_2$. Otherwise, $r' \leq R_3$ and

$$\begin{aligned} d(j, i_2) &\leq d(j, i') + d(i', i'') + d(i'', i_1) + d(i_1, i_3) + d(i_3, i_2) \\ &\leq 3 \cdot R_3 + R_3 + R_1 + R_1 + R_2 \\ &= 2 \cdot R_1 + R_2 + 4 \cdot R_3. \end{aligned}$$

So radius $C^{(2)} := \max\{3 \cdot R_2, 2 \cdot R_1 + R_2 + 4 \cdot R_3\}$ suffices if we choose i_2 as the center.

- **Centering at i_3 .** If $(i', r') = (i_2, R_2)$ then $d(j, i_3) \leq d(j, i_2) + d(i_2, i_3) \leq 3 \cdot R_2 + R_2 = 4 \cdot R_2$. Otherwise, $r' \leq R_3$ and we see

$$\begin{aligned} d(j, i_3) &\leq d(j, i') + d(i', i'') + d(i'', i_1) + d(i_1, i_3) \\ &\leq 3 \cdot R_3 + R_3 + R_1 + R_1 \\ &= 2 \cdot R_1 + 4 \cdot R_3. \end{aligned}$$

So radius $C^{(3)} := \max\{4 \cdot R_2, 2 \cdot R_1 + 4 \cdot R_3\}$ suffices if we choose i_3 as the center.

With these bounds, we now describe how to choose a single ball covering the points covered by tripled balls in $G_{i,r}$ in a way that gives a good bound on the minimum-radius ball covering these points. The following cases employ particular constants to decide which center should be used, these have been optimized for our approach. The final bounds are stated to be of the form $3 \cdot C_{i,r}$ plus some multiple of r . Let $C_{i,r} = \sum_{(i',r') \in G_{i,r}} r$ be the total radii of all balls in $G_{i,r}$. So $\sum_{G_{i,r} \in \mathcal{G}} C_{i,r} = \text{cost}(\mathcal{B}_1) = C_1$.

■ **Case:** $R_3 > R_2/3$.

Then the ball $B'_{i,r}$ is selected to be $B(i_1, C^{(1)})$. Note $4/3 \cdot R_2 < R_2 + R_3 \leq C_{i,r}$ so $C^{(1)} \leq r + 3 \cdot C_{i,r}$.

■ **Case:** $R_3 \leq R_2/3$ and $R_2 \geq \frac{6}{5} \cdot R_1$.

The ball $B'_{i,r}$ is selected to be $B(i_2, C^{(2)})$. Note $C^{(2)} \leq \frac{6}{5} \cdot R_1 + 3 \cdot R_2 \leq \frac{6}{5} \cdot r + 3 \cdot C_{i,r}$.

■ **Case:** $R_3 \leq R_2/3$ and $\frac{6}{5} \cdot R_1 > R_2 \geq \frac{3}{8} \cdot R_1$.

The ball $B'_{i,r}$ is selected to be $B(i_3, C^{(3)})$. Note $C^{(3)} \leq \frac{11}{8} \cdot R_1 + 3 \cdot R_2 \leq \frac{11}{8} \cdot r + 3 \cdot C_{i,r}$.

■ **Case:** $R_3 \leq R_2/3$ and $\frac{3}{8} \cdot R_1 > R_2$.

The ball $B'_{i,r}$ is selected to be $B(i_1, C^{(1)})$. Note $C^{(1)} \leq \frac{11}{8} \cdot R_1 + 3 \cdot R_2 \leq \frac{11}{8} \cdot r + 3 \cdot C_{i,r}$.

In any case, we see that by selecting $B'_{i,r}$ optimally, the radius is at most $\frac{11}{8} \cdot r + 3 \cdot C_{i,r}$. Also, since $R_1, R_2, R_3 \leq 3 \cdot R_m$ by Theorem 6, then the radius of $B'_{i,r}$ is also seen to be at most, say, $21 \cdot R_m$.

Choosing which groups to merge

For each group $G_{i,r} \in \mathcal{G}$, we consider two options. Either we select all balls in $G_{i,r}$ with triple their original radii (thus, with total cost $3 \cdot C_{i,r}$), or we select the single ball $B'_{i,r}$ described in the previous section. We want to do this to minimize the resulting cost while ensuring the number of centers open is at most k' . To help with this, we consider the following linear program. For each $G_{i,r} \in \mathcal{G}$ we have a variable $z_{i,r}$ where $z_{i,r} = 0$ corresponds to selecting the $|G_{i,r}|$ balls with triple their original radius and $z_{i,r} = 1$ corresponds to selecting the single ball $B'_{i,r}$. As noted in the previous section, the radius of $B'_{i,r}$ is at most $\frac{11}{8} \cdot r + 3 \cdot C_{i,r}$ and also at most $21 \cdot R_m$.

$$\begin{aligned} \text{minimize : } & \sum_{G_{i,r} \in \mathcal{G}} (1 - z_{i,r}) \cdot 3 \cdot C_{i,r} + z_{i,r} \cdot \text{cost}(\{B'_{i,r}\}) \\ \text{subject to : } & \sum_{G_{i,r} \in \mathcal{G}} ((1 - z_{i,r}) \cdot |G_{i,r}| + z_{i,r}) \leq k' \\ & z_{i,r} \in [0, 1] \quad \forall G_{i,r} \in \mathcal{G} \end{aligned} \quad (\text{LP-Choose})$$

To consolidate the groups, compute an optimal extreme point to **LP-Choose**. Since all but one constraint are $[0, 1]$ box constraints, there is at most one variable $z_{i,r}$ that does not take an integer value. Since $|G_{i,r}| \geq 1$, then setting $z_{i,r}$ to 1 yields a feasible solution whose cost increases by at most the radius of $B'_{i,r}$, which was observed to be at most $21 \cdot R_m \leq 21 \cdot \epsilon \cdot \text{OPT}$.

Recall that a, b are such that $a, b \geq 0, a + b = 1$ and $a \cdot k_1 + b \cdot k_2 = k'$. Setting $z_{i,r} = a$ for each $G_{i,r} = 1$ is feasible since $1 - z_{i,r} = b, \sum_{G_{i,r} \in \mathcal{G}} |G_{i,r}| = k_2$, and there are at most k'_1 terms in this sum. The value of this solution is

$$\sum_{G_{i,r} \in \mathcal{G}} (3 \cdot b + 3 \cdot a) \cdot C_{i,r} + \frac{11}{8} \cdot b \cdot r = 3 \cdot C_2 + \frac{11}{8} \cdot b \cdot C_1$$

so the optimum solution to **LP-Choose** has value at most this as well. Summarizing,

► **Lemma 7.** *In polynomial time, we can compute a set of at most k' balls with total radius at most $\frac{11}{8} \cdot b \cdot C'_1 + 3 \cdot C_2 + 21 \cdot \epsilon \cdot OPT$ which cover all points in X' .*

Finally, we can complete our analysis. Recall our simple solution of tripling the balls in \mathcal{B}'_1 has cost at most $3 \cdot C'_1$ and the more involved solution just described has cost at most $3 \cdot C_1 + \frac{11}{8} \cdot a \cdot C_2 + 21 \cdot \epsilon \cdot OPT$. Now,

$$\min \left\{ 3 \cdot C_2, 3 \cdot C_1 + \frac{11}{8} \cdot b \cdot C_2 \right\} \leq (1-d) \cdot 3 \cdot C_2 + d \cdot \left(b \cdot \frac{11}{8} \cdot C_2 + 3 \cdot C_1 \right)$$

holds for any $0 \leq d \leq 1$. To maximize the latter, we set $d = \frac{3(1-b)}{\frac{11}{8} \cdot b^2 - \frac{11}{8} \cdot b + 3}$ and see the minimum of these two terms is at most

$$\left(\frac{9}{\frac{11}{8} \cdot b^2 - \frac{11}{8} \cdot b + 3} \right) \cdot (aC_1 + bC_2) \leq \left(\frac{9}{\frac{11}{8} \cdot b^2 - \frac{11}{8} \cdot b + 3} \right) \cdot OPT'$$

where we have used bound 1 for the last step.

The worst case occurs at $b = \frac{1}{2}$, at which the bound becomes $85/288 \cdot OPT'$. Thus, the cost of the solution is at most $\frac{288}{85} \cdot OPT' + 21 \cdot \epsilon \cdot OPT$. Adding the balls \mathcal{B}' we guessed to also cover the points in $X - X'$, we get a solution covering all of X with total radii at most

$$\text{cost}(\mathcal{B}') + \frac{288}{85} \cdot OPT' + 21 \cdot \epsilon = OPT - OPT' + \frac{288}{85} \cdot OPT' + 21 \cdot \epsilon \cdot OPT \leq 3.389 \cdot OPT$$

for sufficiently small ϵ .

Algorithm Summary

The entire algorithm for MSR that we have just presented is summarized in Algorithm 1.

■ Algorithm 1 MSR Approximation.

$\mathcal{S} \leftarrow \emptyset$ {The set of all solutions seen over all guesses}

for each subset \mathcal{B}'_j of $1/\epsilon$ balls **do**

 let X', R_m be as described in Section 2.2

$(\mathcal{A}, R) \leftarrow k$ -MEDIAN 2-approximation on X'

if $R > 2 \cdot R_m$ **then**

reject this guess \mathcal{B}' and continue with the next

 let $\mathcal{B}_1, \mathcal{B}_2, \lambda$ be the bipoint solution described in Theorem 6

 let \mathcal{G} be the groups (a partitioning of \mathcal{B}_1) described in Section 2.4

 for each $G_{i,r} \in \mathcal{G}$, let $B'_{i,r}$ be the cheapest ball covering $\cup_{(i',r') \in G_{i,r}} B(i', 3 \cdot r')$

 let z' be an optimal extreme point to **LP-Choose**

$\mathcal{B}^{(1)} \leftarrow \{B'_{i,r} : z'_{i,r} > 0\} \cup \cup_{z'_{i,r}=0} \{(i', 3 \cdot r') : (i', r') \in G_{i,r}\}$

$\mathcal{B}^{(2)} \leftarrow \{(i, 3 \cdot r) : (i, r) \in \mathcal{B}_2\}$

 let \mathcal{B} be $\{(i, 3 \cdot r) : (i, r) \in \mathcal{B}'\}$ plus the cheaper of the two sets $\mathcal{B}^{(1)}$ and $\mathcal{B}^{(2)}$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathcal{B}\}$

return the cheapest solution from \mathcal{S}

3 Minimum Sum of Diameters

Here, we observe that a slight modification to the MSR approximation in fact yields a 6.546-approximation for MSD. Note that for any $Y \subseteq X$ with diameter, say, $\text{diam}(Y)$, for any $i \in Y$ we have $Y \subseteq B(i, \text{diam}(Y))$ and $\text{diam}(B(i, \text{diam}(Y))) \leq 2 \cdot \text{diam}(Y)$. So while it is difficult to guess any single cluster from the optimum MSD solution, we can guess the $1/\epsilon$ largest diameters (the values) and guess balls \mathcal{B}' with these radii that cover these largest-diameter clusters. Let OPT'_D denote the total diameter of the remaining clusters from the optimum solution, $k' = k - \frac{1}{\epsilon}$, X' be the remaining points to cluster, and $R_m = \min\{r : (i, r) \in \mathcal{B}'\} \leq \epsilon \cdot \text{OPT}'_D$.

For any $\lambda \geq 0$, note $\text{OPT}'_{\text{LP}(\lambda)} + \lambda \cdot k' \leq \text{OPT}'_D$ as picking any single center from each cluster in optimum solution on X' yields an MSR solution with cost at most OPT'_D . We then use Theorem 6 to get a bipoint solution $\mathcal{B}_1, \mathcal{B}_2, \lambda$.

If we triple the balls in \mathcal{B}_2 and output those clusters, we get a solution with total diameter $\leq 6 \cdot \text{cost}(\mathcal{B}_2)$. For the other case, we again form groups \mathcal{G} . Instead of picking a ball $B'_{i,r}$ for each group $G_{i,r} \in \mathcal{G}$, we simply let $B'_{i,r}$ be the set of points covered by the tripled balls in $G_{i,r}$. We claim $\text{diam}(B'_{i,r}) \leq 2 \cdot r + 6 \cdot C_{i,r}$.

To see this, consider any two points j', j'' covered by $\cup_{(i', r') \in G_{i,r}} B(i', 3 \cdot r')$, say (i', r') and (i'', r'') are the balls in $G_{i,r}$ which, when tripled, cover j' and j'' , respectively. If $(i', r') = (i'', r'')$ (i.e. it is the same tripled ball from $G_{i,r}$ that covers both j', j'') then $d(j', j'') \leq 6 \cdot r' \leq 6 \cdot C_{i,r}$. Otherwise, we have $r' + r'' \leq C_{i,r}$ and

$$d(j', j'') \leq d(j', i') + d(i', i'') + d(i'', i'') + d(i'', j'') \leq 4 \cdot r' + r + r + 4 \cdot r'' \leq 2 \cdot r + 4 \cdot C_{i,r}.$$

In either case, we can upper bound $d(j', j'') \leq 2 \cdot r + 6 \cdot C_{i,r}$, so $\text{diam}(B'_{i,r})$ is bounded by the same. We use an LP similar to **LP-Choose** except with the modified objective function to reflect the diameter costs of the corresponding choices.

$$\begin{aligned} \text{minimize : } & \sum_{G_{i,r} \in \mathcal{G}} (1 - z_{i,r}) \cdot 6 \cdot C_{i,r} + z_{i,r} \cdot \text{diam}(B'_{i,r}) \\ \text{subject to : } & \sum_{G_{i,r} \in \mathcal{G}} ((1 - z_{i,r}) \cdot |G_{i,r}| + z_{i,r}) \leq k' \\ & z_{i,r} \in [0, 1] \quad \forall G_{i,r} \in \mathcal{G} \end{aligned} \quad (\text{LP-Choose MSD})$$

For $a, b \geq 0$, we let $a + b = 1$ and $a \cdot k_1 + b \cdot k_2 = k'$, similar to MSR. Setting $z_{i,r} = a$ shows the optimum LP solution value is at most

$$\sum_{G_{i,r} \in \mathcal{G}} (6 \cdot b + 6 \cdot a) \cdot C_{i,r} + 2 \cdot b \cdot r = 6 \cdot C_2 + 2 \cdot b \cdot C_1.$$

In an optimal extreme point, at most one variable in **LP-Choose MSD** that is fractional so we set it to 1 we pick to corresponding group to be covered by a single ball. The final cost is $\min\{6 \cdot C_2, 6 \cdot C_1 + 2 \cdot b \cdot C_2 + O(\epsilon) \cdot \text{OPT}'_D\} \leq (1 - d) \cdot 6 \cdot C_2 + d \cdot (b \cdot 2 \cdot C_2 + 6 \cdot C_1)$ for any $d \in [0, 1]$. Let At $d = \frac{6(1-b)}{2 \cdot b^2 - 2 \cdot b + 6}$ the worst case for the final bound is at $b = 1/2$ at which we see the cost is at most $\frac{72}{11} \cdot \text{OPT}'_D + O(\epsilon) \cdot \text{OPT}'_D$. Adding this to the $1/\epsilon$ balls we guessed (whose diameters are at most twice their radius) and choosing ϵ sufficiently small shows we get a solution with an approximation guarantee of 6.546 for MSD, which is better than two times the MSR guarantee.

4 Getting the Bipoint Solution: Proof of Theorem 6

We again emphasize that one can slightly adapt the algorithm and analysis in [7] to prove a slightly weaker version of Theorem 6 that would still suffice for our approximation guarantees. The main difference is that the averaging of the bipoint solution costs as given in bound (1) from Section 2.4 would be bounded by $(1 + \epsilon') \cdot OPT$ for some $\epsilon' > 0$ (the running time depends linearly on $\log 1/\epsilon$).

We give an alternative approach that uses simple LP rounding. This may be of independent interest since our method of getting a single λ rather than two “close” values λ_1, λ_2 is simple in principle and may apply to other Lagrangian multiplier preserving (LMP) approximations that use direct LP rounding. That is, we give a recipe to find a single λ and two corresponding solutions $\mathcal{B}_1, \mathcal{B}_2$ that uses very generic properties of the rounding algorithm.

The proof of Theorem 6 proceeds through the usual approach of using a binary search using an LMP algorithm. We begin by describing our LMP algorithm followed by a simple consolidation step which is used in some parts of the binary search. Our direct LP rounding procedure is presented here as is an outline of the binary search routine. For the sake of space, full details of how our binary search works with the rounding procedure are deferred to full version.

4.1 A Simple LMP Algorithm via Direct LP Rounding

Algorithm 2 describes our rounding procedure. Note it only depends on x' and not on λ itself.

■ **Algorithm 2** ROUND(x').

```

 $\mathcal{B} \leftarrow \emptyset$ 
for  $(i, r)$  with  $x'_{i,r} > 0$  in non-increasing order of  $r$  do
  if  $B(i, r) \cap B(i', r') = \emptyset$  for each  $(i', r') \in \mathcal{B}$  then
     $\mathcal{B} \leftarrow \mathcal{B} \cup \{(i, r)\}$ 
return  $\mathcal{B}$ 

```

To analyze the performance of this algorithm, we also consider the dual of $\mathbf{LP}(\lambda)$.

$$\begin{array}{ll}
 \max & \sum_{j \in X'} y_j \\
 \text{s.t.} & \sum_{j \in B(i,r) \cap X'} y_j \leq r + \lambda \quad \forall (i, r), r \leq R_m \\
 & y \geq 0
 \end{array} \quad (\mathbf{DUAL}(\lambda))$$

► **Theorem 8.** *Let $\lambda \geq 0$. Let x' be an optimal solution for $\mathbf{LP}(\lambda)$ and y' be an optimal dual solution for $\mathbf{LP}(\lambda)$. Let \mathcal{B} denote the set returned by ROUND(x'). The balls in \mathcal{B} are pairwise-disjoint and for each $(i, r) \in \mathcal{B}$ we have $r \leq R_m$ and $r + \lambda = \sum_{j \in B(i,r)} y'_j$.*

Proof. Disjointedness follows by construction. Each ball \mathcal{B} has radius at most R_m since each ball is from the support of x' and $\mathbf{LP}(\lambda)$ only has variable for balls with radius $\leq R_m$. Again, since each $(i, r) \in \mathcal{B}$ lies in the support of x' then complementary slackness shows $r + \lambda = \sum_{j \in B(i,r)} y'_j$. ◀

Note the last condition shows $\text{cost}(\mathcal{B}) + \lambda \cdot |\mathcal{B}| \leq \sum_{j \in X'} y'_j = OPT_{\mathbf{LP}(\lambda)}$. Thus, we call this a “Lagrangian multiplier preserving” algorithm because if \mathcal{B}'' is obtained by tripling the radii of the balls returned by ROUND(x'), then $\text{cost}(\mathcal{B}'') + 3 \cdot \lambda \cdot |\mathcal{B}''| \leq 3 \cdot OPT_{\mathbf{LP}(\lambda)}$.

4.2 Sketch of the Binary Search

While it is fairly easy to see that using $\lambda = 0$ will have an optimal LP solution be rounded to $|X'| > k'$ balls, we need to ensure that for large enough λ that our rounding procedure produces $\leq k'$ balls in order to begin our binary search. Thus, we consider another step $\text{CONSOLIDATE}(\mathcal{B}, \lambda, \mathcal{A}, R_m)$ that tries to consolidate some of the balls output by $\text{ROUND}(x')$ using the balls from the k -MEDIAN approximation \mathcal{A} . Roughly speaking, if the single radius $3 \cdot R_m$ -ball centered at some $i' \in \mathcal{A}'$ is cheaper than the balls of \mathcal{B} it covers, we replace them with this single ball.

■ **Algorithm 3** $\text{CONSOLIDATE}(\mathcal{B}, \lambda, \mathcal{A}, R_m)$.

```

 $\mathcal{B}^c \leftarrow \emptyset$ 
for each  $i' \in \mathcal{A}$  do
  Let  $N_{i'} = \{(i, r) \in \mathcal{B} : i \in B(i', 2 \cdot R_m)\}$ 
  if  $3 \cdot R_m + \lambda \leq \sum_{(i,r) \in N_{i'}} (r + \lambda)$  then
     $\mathcal{B}^c \leftarrow \mathcal{B}^c \cup \{(i', 3 \cdot R_m)\}$ 
     $\mathcal{B} \leftarrow \mathcal{B} - N_{i'}$ 
 $\mathcal{B}^c \leftarrow \mathcal{B}^c \cup \mathcal{B}$ 
return  $\mathcal{B}^c$ 

```

We the following show in the full version.

► **Lemma 9.** *Let $\mathcal{B} = \text{ROUND}(x')$ for some optimal solution x' for $\text{LP}(\lambda)$ and $\mathcal{B}^c = \text{CONSOLIDATE}(\mathcal{B}, \lambda, \mathcal{A}, R_m)$.*

1. *If $\lambda \geq 4 \cdot R_m$, then $|\mathcal{B}^c| \leq k'$,*
2. *$r \leq 3 \cdot R_m$ for each $(i, r) \in \mathcal{B}^c$*
3. *for each $(i, r) \in \mathcal{B}^c$ there is some $X_{i,r} \subseteq B(i, r)$ such that $r + \lambda \leq \sum_{j \in X_{i,r}} y'_j$ where y' is an optimal solution to $\text{DUAL}(\lambda)$,*
4. *for different $(i, r), (i', r') \in \mathcal{B}^c$ we have $X_{i,r} \cap X_{i',r'} = \emptyset$, and*
5. *for each $j \in \mathcal{X}'$ we have $j \in B(i, 3 \cdot r)$ for some $(i, r) \in \mathcal{B}^c$.*

In particular, the total $(r + \lambda)$ -cost of \mathcal{B}^c is still at most $\text{OPT}_{\text{LP}(\lambda)}$ since properties 3 and 4 show each ball can be paid for some variables in the optimal dual solution and no variable in the dual is charged more than once this way.

In this way, we can start the binary search with $\lambda_1 = 0$ (for which ROUND will return exactly k' balls) and $\lambda_2 = 4 \cdot R_m$ (for which consolidating the rounded solution will produce $\leq k'$ balls). During the binary search, if at any step the optimal LP solution x_1 to $\text{LP}(\lambda_1)$ is also an optimal LP solution to $\text{LP}(\lambda_2)$ (here $[\lambda_1, \lambda_2]$ is the current interval enclosed by the binary search) or if $|\mathcal{B}| \geq k' \geq |\mathcal{B}^c|$ where \mathcal{B} is obtained by rounding an optimal solution to $\text{LP}(\lambda)$ and \mathcal{B}^c is obtained by consolidating it, it is easy to find the bipoint solution satisfying Theorem 6.

So we focus on *break points* λ which, intuitively, are values λ where the set of optimal extreme points to $\text{LP}(\lambda)$ changes. In the full version, we prove there is sufficiently-large distance between distinct breakpoints. So after a polynomial number of iterations, if the search did not terminate for one of the reasons mentioned above, then the window $[\lambda_1, \lambda_2]$ encloses exactly one break point.

We also show how to compute the largest λ such that x_1 remains optimal for $\text{LP}(\lambda)$ by solving yet another a linear program that exploits complementary slackness. After computing the only breakpoint in our final binary search window, it is easy to construct the bipoint solution satisfying the requirements of Theorem 6. We note that whenever we return a bipoint solution from this binary search, we consider a post-processing routine to ensure each ball in \mathcal{B}_1 will intersect at least one ball in \mathcal{B}_2 to fulfill the requirements of Theorem 6.

5 Concluding Remarks

It may be possible to improve our analysis further by considering an even more involved approach to analyzing how to optimally cover a group using a single ball, though such an approach seems likely to produce approximations that are still a constant-factor worse than 3. What is more interesting is an observation about our new approach to finding the bipoint solution. If we ever encounter a λ such that $|\mathcal{B}| \geq k' \geq |\mathcal{B}^c|$ where \mathcal{B} is the output of ROUND and \mathcal{B}^c is the output of CONSOLIDATE (using \mathcal{B}), then Check 1 will terminate the search with bipoint solution $\mathcal{B}, \mathcal{B}^c$. If we refine the CONSOLIDATE step to perform the consolidations for \mathcal{B} one at a time and stop when the number of clusters first becomes $\leq k'$, one can show that tripling the radii of these $\leq k'$ balls is a solution with cost at most $(3 + O(\epsilon)) \cdot OPT'$. But this is just for one case in our binary search. In general, is there a refinement of our binary search routine (or some other approach) that would always produce a $(3 + \epsilon)$ -approximation?

References

- 1 N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-ptas for unsplittable flow on line graphs. In *Proceedings of 38th Annual Symposium on Theory of Computing (STOC)*, STOC '06, pages 721–729, New York, NY, USA, 2006. Association for Computing Machinery.
- 2 J. Batra, N. Garg, A. Kumar, T. Mömke, and A. Wiese. New approximation schemes for unsplittable flow on a path. In *Proceedings of the 2015 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 47–58, 2015.
- 3 B. Behsaz and M. R. Salavatipour. On minimum sum of radii and diameters clustering. *Algorithmica*, 73(1):143–165, September 2015.
- 4 V. Capoteas, G. Rote, and G. Woeginger. Geometric clusterings, 1990.
- 5 M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 378–388, 1999.
- 6 M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.
- 7 M. Charikar and R. Panigrahy. Clustering to minimize the sum of cluster diameters. *Journal of Computer and System Sciences*, 68(2):417–441, 2004. Special Issue on STOC 2001.
- 8 S. Doddi, M. V. Marathe, S. S. Ravi, D. S. Taylor, and P. Widmayer. Approximation algorithms for clustering to minimize the sum of diameters. In *In Proceedings of 7th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 237–250. Springer-Verlag, 2000.
- 9 M. E. Dyer and A. M. Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3(6):285–288, 1985.
- 10 M. Gibson, G. Kanade, E. Krohn, I. A. Pirwani, and K. Varadarajan. On metric clustering to minimize the sum of radii. *Algorithmica*, 57(3):484–498, February 2009.
- 11 M. Gibson, G. Kanade, E. Krohn, I. A. Pirwani, and K. Varadarajan. On clustering to minimize the sum of radii. *SIAM Journal on Computing*, 41(1):47–60, 2012.
- 12 F. Grandoni, T. Mömke, and A. Wiese. A ptas for unsplittable flow on a path. In *To appear in proceedings of 54th ACM Symposium on Theory of Computing (STOC)*, 2022.
- 13 F. Grandoni, T. Mömke, and A. Wiese. Unsplittable flow on a path: The game! In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 906–926, 2022.
- 14 F. Grandoni, T. Mömke, A. Wiese, and A. Zhou. To augment or not to augment: Solving unsplittable flow on a path by creating slack. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2411–2422, 2017.

- 15 F. Grandoni, T. Mömke, A. Wiese, and H. 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, New York, NY, USA, 2018. Association for Computing Machinery.
- 16 P. Hansen and B. Jaumard. Minimum sum of diameters clustering. *Journal of Classification*, 4(2):215–226, September 1987.
- 17 P. Heggernes and D. Lokshtanov. Optimal broadcast domination in polynomial time. *Discrete Mathematics*, 306(24):3267–3280, 2006.
- 18 D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k -center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.
- 19 K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k -median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, March 2001.
- 20 O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. ii: The p -medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560, 1979.
- 21 J. H. Lin and J. S. Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44(5):245–249, 1992.

Simple Worst-Case Optimal Adaptive Prefix-Free Coding

Travis Gagie  

Faculty of Computer Science, Dalhousie University, Halifax, Canada

Abstract

We give a new and simple worst-case optimal algorithm for adaptive prefix-free coding that matches Gagie and Nekrich’s (2009) bounds except for lower-order terms, and uses no data structures more complicated than a lookup table.

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

Keywords and phrases Adaptive prefix-free coding, Shannon coding, Lookup tables

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.57

Funding *Travis Gagie*: Funded by NSERC Discovery Grant RGPIN-07185-2020.

1 Introduction

Suppose Alice has a deck of n cards, each marked with a character from a known alphabet of size σ , and she wants to send the sequence of the cards’ characters to Bob over a noiseless binary channel. Moreover, suppose neither of them know in advance the frequencies of the distinct characters in the deck – perhaps it has just been shuffled – and Alice wants to encode each card’s character before looking at the next card, such that Bob can recognize the last bit of that character’s encoding when he receives it, and decode the character.

In 1973 Faller [1] proposed that Alice encode each card’s character with a Huffman code [7] for the distribution of characters she has already seen, modified to assign codewords also to characters she has not yet seen. Thus, Alice encodes the i th character using a code that depends only on the first $i - 1$ characters. Assuming Bob has already decoded the first $i - 1$ characters when he receives the encoding of the i th, he can build the same code. Because the codewords in a Huffman code are prefix-free – no codeword is a prefix of another – he can then use that code to decode the i th character when he receives the last bit of its encoding. Since the codes adapt to the frequencies of the characters as more and more are seen, Faller’s algorithm is said to perform adaptive Huffman coding or, more generally, *adaptive prefix-free coding*.

Building n Huffman codes from scratch takes $\Omega(n\sigma)$ time, but Faller showed how Alice and Bob can store a Huffman code such that after incrementing the frequency of a character, they can update the code in time proportional to the length of the codeword previously assigned to that character. It follows that Faller’s algorithm runs in time proportional to the total length of the encoding of the sequence. Gallager [6] and Knuth [11] further developed Faller’s algorithm in 1978 and 1985, respectively, and it is usually known as the FGK algorithm for their initials. It is commonly taught in courses on data compression and used in the classic UNIX utility `compact`, for example.

The same year Knuth published his work on the FGK algorithm, Vitter [15, 16, 17] gave a more sophisticated algorithm for adaptive Huffman coding. He showed it uses less than 1 more bit per character than Alice would use if she knew the characters’ frequencies in advance, built a single Huffman code for them, sent it to Bob, and then used it to encode and send the sequence of characters in the deck. In other words, Vitter’s algorithm uses at most about $n(H + 1 + \delta)$ bits and $O(n(H + 1))$ time overall, where



© Travis Gagie;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 57; pp. 57:1–57:5



Leibniz International Proceedings in Informatics

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

$$H = \sum_{j=1}^{\sigma} \frac{n_j}{n} \lg \frac{n}{n_j} \leq \lg \sigma$$

is the entropy of the distribution $\frac{n_1}{n}, \dots, \frac{n_{\sigma}}{n}$ of characters in the deck and $\delta \in [0, 1)$ is the redundancy of a Huffman code for that distribution. (When the distribution is dyadic – each frequency n_i is n divided by a power of 2 – then $\delta = 0$, and when nearly all the cards have the same character then δ is nearly 1.) Vitter’s algorithm is also commonly taught in courses on data compression.

Vitter attributed to Chazelle an observation that the FGK algorithm uses at most about twice as many bits as using a single Huffman code for the characters’ frequencies (the coefficient 2 can be reduced to about 1.44 using a result by Katona and Nemetz [10]), so the FGK algorithm also runs in $O(n(H + 1))$ time. In 1999 Milidiú, Laber and Pessoa [12] showed that with the FGK algorithm, Alice sends fewer than 2 more bits per character than she would with a single Huffman code for the characters’ frequencies, or at most about $n(H + 2 + \delta)$ bits overall.

In 2003 Gagie [3, 4] showed that if we modify the FGK algorithm to perform adaptive Shannon coding instead of adaptive Huffman coding, then Alice sends at most about $H + 1$ bits per character. A Shannon code [13] is a prefix-free code that assigns any character with probability p a codeword of length at most $\lceil \lg \frac{1}{p} \rceil$ so, if Alice pretends she has seen each character once before she starts encoding the sequence $S[1..n]$ of the cards’ characters, then she sends at most

$$\begin{aligned} & \sum_{i=1}^n \left\lceil \lg \frac{i + \sigma - 1}{\text{occ}(S[i], S[1..i - 1]) + 1} \right\rceil \\ & \leq \sum_{i=1}^n \lg(i + \sigma - 1) - \sum_{i=1}^n \lg(\text{occ}(S[i], S[1..i - 1]) + 1) + n \\ & < \lg n! + \sigma \lg(n + \sigma) - \sum_{j=1}^{\sigma} \lg n_j! + n \\ & = \lg \binom{n}{n_1, \dots, n_{\sigma}} + \sigma \lg(n + \sigma) + n \end{aligned}$$

bits, where $\text{occ}(S[i], S[1..i - 1])$ is the frequency of $S[i]$ in the prefix $S[1..i - 1]$ and $\binom{n}{n_1, \dots, n_{\sigma}}$ is the number of distinct ways of arranging the cards in the deck. By Stirling’s Approximation, $\lg \binom{n}{n_1, \dots, n_{\sigma}} \leq nH + O(\sigma \log n)$, so with Gagie’s algorithm Alice sends $n(H + 1) + o(n)$ bits as long as $\sigma \lg(n + \sigma) \in o(n)$.

In 2006 Karpinski and Nekrich [8, 9] gave a more sophisticated algorithm for adaptive Shannon coding, with which Alice sends $n(H + 1) + O(\sigma \log^2 n)$ bits and encodes S in $O(n)$ time, and Bob decodes it in $O(n(\log H + 1))$ time. Their algorithm uses canonical codes [14] and assumes Alice and Bob are working on word RAMs with $\Omega(\log n)$ -bit words, as we do henceforth. Notice speeding up encoding on a word RAM is generally easier than speeding up decoding: for example, given a single prefix-free code whose longest codeword fits in a constant number of machine words, Alice can build an $O(\sigma)$ -space lookup table that tells her the codeword for any character in constant time.

In 2009 Gagie and Nekrich [5] improved Karpinski and Nekrich’s algorithm so that Bob decodes S also in $O(n)$ time, at the cost of increasing the bound on the total encoding length to $n(H + 1) + O(\sigma \log^{5/2} n)$. They also proved Alice must send $n(\lg \sigma + 1 - o(1)) \geq$

■ **Table 1** The per-character bounds for the algorithms discussed, assuming $\sigma \in o\left(\frac{n^{1/2}}{\log n}\right)$, ignoring lower-order terms and omitting asymptotic notation.

authors	encoding length	encoding time	decoding time
FGK [1, 6, 11]	$H + 2 + \delta$	$H + 1$	$H + 1$
Vitter [15, 16, 17]	$H + 1 + \delta$	$H + 1$	$H + 1$
Gagie [3, 4]	$H + 1$	$H + 1$	$H + 1$
KN [8, 9]	$H + 1$	1	$\lg H + 1$
GN [5]	$H + 1$	1	1
new	$H + 1$	1	1

$n(H + 1 - o(1))$ bits in the worst case, even when even when $\sigma \in \omega(1)$. To see why, suppose $\sigma = 2^{\lceil \lg f(n) \rceil} + 1$ for some function $f(n) \in \omega(1)$, so $\sigma \in \omega(1)$ and any prefix-free code for the alphabet assigns some character a codeword of length $\lceil \lg f(n) \rceil + 1 = \lg \sigma + 1 - o(1)$. For each i , the adversary chooses $S[i]$ to be a character with codeword length at least $\lg \sigma + 1 - o(1)$ in the code Alice will use to encode $S[i]$.

Gagie and Nekrich's algorithm is simultaneously worst-case optimal in terms encoding and decoding time and of encoding length, as long as $\sigma \in o\left(\frac{n}{\log^{5/2} n}\right)$, but it relies on constant-time predecessor queries on sets of $O(\log^{1/6} n)$ elements. These are theoretically possible on a word RAM with $\Omega(\log n)$ -bit words [2] but very complicated and totally impractical. In this paper we give a new algorithm for adaptive prefix-free coding that is simple – it uses no data structures more complicated than a lookup table – but still uses $O(n)$ time for both encoding and decoding and $n(H + 1) + O\left(\frac{n}{\log n} + \sigma^2 \log^2 n\right)$ bits, which is within lower-order terms of optimal when $\sigma \in o\left(\frac{n^{1/2}}{\log n}\right)$. Table 1 shows the per-character bounds of all the algorithms we have discussed here, assuming $\sigma \in o\left(\frac{n^{1/2}}{\log n}\right)$, ignoring lower-order terms and omitting asymptotic notation. We leave as future work finding a simple algorithm that is worst-case optimal even when σ is closer to n .

2 Algorithm

Before we describe our new algorithm, we briefly review how length-restricting a prefix-free code can speed up decoding. Our starting point is Gagie's [3, 4] observation that if we smooth the probability distribution $\frac{n_1}{n}, \dots, \frac{n_\sigma}{n}$ by averaging it with the uniform distribution and apply Shannon's construction [13] to the result, then we obtain a prefix-free code with average codeword length

$$\sum_{j=1}^{\sigma} \frac{n_j}{n} \left\lceil \lg \frac{1}{\frac{1}{2} \left(\frac{n_j}{n} + \frac{1}{\sigma} \right)} \right\rceil < \sum_{j=1}^{\sigma} \frac{n_j}{n} \left(\lg \frac{n}{n_j} + 2 \right) = H + 2$$

when encoding S , and maximum codeword length at most $\lceil \lg \sigma \rceil + 1$.

In $O(\sigma)$ time we can build an $O(\sigma)$ -space lookup table that, for any binary string of length $\lceil \lg \sigma \rceil + 1$, tells us which character's codeword is a prefix of that string and the length of that codeword. If we encode S by replacing each character by its codeword – which we can do in $O(n)$ time using an $O(\sigma)$ -space lookup table that tells us the codeword for any character – then later we can decode S in $O(n)$ time by repeatedly taking prefixes of the encoding consisting $\lceil \lg \sigma \rceil + 1$ bits, looking up which character's codeword is a prefix of the encoding and the length of that codeword, and deleting the codeword from the beginning of the encoding. Unfortunately, we may use about $H + 2$ bits per character.

If we take a weighted average of $\frac{n_1}{n}, \dots, \frac{n_\sigma}{n}$ and the uniform distribution, however, then we can reduce the number of bits we use per character, at the cost of increasing the maximum codeword length and the space needed by the table. For example, if we assign weight $\frac{\lg n - 1}{\lg n}$ to $\frac{n_1}{n}, \dots, \frac{n_\sigma}{n}$ and weight $\frac{1}{\lg n}$ to the uniform distribution before averaging them and applying Shannon's construction, then the average codeword length when encoding S is

$$\sum_{j=1}^{\sigma} \frac{n_j}{n} \left[\lg \frac{1}{\frac{\lg n - 1}{\lg n} \cdot \frac{n_j}{n} + \frac{1}{\lg n} \cdot \frac{1}{\sigma}} \right] < \sum_{j=1}^{\sigma} \frac{n_j}{n} \left(\lg \frac{n}{n_j} + \lg \frac{\lg n}{\lg n - 1} + 1 \right) < H + 1 + \frac{\lg e}{\lg n - 1}$$

and the maximum codeword length is at most $\lceil \lg(\sigma \lg n) \rceil = \lceil \lg \sigma + \lg \lg n \rceil$, so the lookup table takes $O(2^{\lceil \lg \sigma + \lg \lg n \rceil}) = O(\sigma \lg n)$ space. Building this table takes $O(\sigma \lg n)$ time.

We are now ready to describe our new algorithm. First, Alice encodes $S[1..\lceil \sigma \lg n \rceil]$ using a Shannon code C_0 for the uniform distribution, that assigns every character a codeword of length $\lceil \lg \sigma \rceil$. This takes her $O(\sigma \lg n)$ time. Then, for $k \geq 1$, after encoding $S[1..k\lceil \sigma \lg n \rceil]$, Alice builds a Shannon code C_k for a weighted average of the distribution of characters she has encoded so far and the uniform distribution:

$$\frac{\lg n - 1}{\ln n} \cdot \frac{\text{occ}(a_1, S[1..k\lceil \sigma \lg n \rceil])}{k\lceil \sigma \lg n \rceil} + \frac{1}{\lg n} \cdot \frac{1}{\sigma}, \dots, \frac{\lg n - 1}{\ln n} \cdot \frac{\text{occ}(a_\sigma, S[1..k\lceil \sigma \lg n \rceil])}{k\lceil \sigma \lg n \rceil} + \frac{1}{\lg n} \cdot \frac{1}{\sigma},$$

where $\text{occ}(a_j, S[1..k\lceil \sigma \lg n \rceil])$ is the frequency in $S[1..k\lceil \sigma \lg n \rceil]$ of the j th character a_j in the alphabet. She builds an $O(\sigma)$ -space lookup table for C_k that lets her encode $S[k\lceil \sigma \lg n \rceil + 1..(k+1)\lceil \sigma \lg n \rceil]$ in $O(\sigma \lg n)$ time.

Because the codewords in C_0 have length $\lceil \lg \sigma \rceil$, Bob can build an $O(\sigma)$ -space lookup table that lets him decode $S[1..\lceil \sigma \lg n \rceil]$ in $O(\sigma \lg n)$ time. Then, for $k \geq 1$, after encoding $S[1..k\lceil \sigma \lg n \rceil]$, Bob builds the same Shannon code C_k that Alice used to encode $S[k\lceil \sigma \lg n \rceil + 1..(k+1)\lceil \sigma \lg n \rceil]$. Because the longest codeword in C_k has length at most $\lceil \lg \sigma + \lg \lg n \rceil$, Bob can build an $O(\sigma \lg n)$ -space lookup table that lets him decode $S[k\lceil \sigma \lg n \rceil + 1..(k+1)\lceil \sigma \lg n \rceil]$ in $O(\sigma \lg n)$ time.

3 Analysis

For each $k \geq 0$, building C_k and the lookup tables for encoding and decoding with it takes Alice and Bob $O(\sigma \lg n)$ time, and that cost is amortized over the $\lceil \sigma \lg n \rceil$ characters in $S[k\lceil \sigma \lg n \rceil + 1..(k+1)\lceil \sigma \lg n \rceil]$. Since encoding and decoding $S[k\lceil \sigma \lg n \rceil + 1..(k+1)\lceil \sigma \lg n \rceil]$ also takes $O(\sigma \lg n)$ time, Alice and Bob each spend $O(n)$ time in total, or constant time per character in S .

Probably the most complicated aspect of our algorithm is the analysis showing the total length of the encoding is at most $n(H + 1) + O\left(\frac{n}{\lg n} + \sigma^2 \log^2 n\right)$. Consider that each character in S is encoded with $O(\sigma \lg n)$ bits so, in particular, the first $\lceil \sigma \lg n \rceil$ occurrences of each distinct character are encoded with a total of $O(\sigma^2 \log^2 n)$ bits. Let I_j be the set of positions i such that character $S[i]$ of S is an occurrence of the j th character a_j in the alphabet but not one of a_j 's first $\lceil \sigma \lg n \rceil$ occurrences. For $i \in I_j$, Alice encodes $S[i]$ using at most

$$\left\lceil \lg \left(\frac{1}{\frac{\lg n - 1}{\lg n} \cdot \frac{i - 1}{\text{occ}(a_j, S[1..i]) - \lceil \sigma \lg n \rceil}} \right) \right\rceil < \lg \frac{i - 1}{\text{occ}(a_j, S[1..i]) - \lceil \sigma \lg n \rceil} + 1 + \frac{\lg e}{\lg n - 1}$$

bits. Therefore, the total number of bits in the encoding is

$$\sum_j \sum \left\{ \lg \frac{i - 1}{\text{occ}(a_j, S[1..i]) - \lceil \sigma \lg n \rceil} : i \in I_j \right\} + n + O\left(\frac{n}{\lg n} + \sigma^2 \log^2 n\right).$$

Notice $\sum_j \sum \{\lg(i-1) : i \in I_j\} \leq \lg n!$ and for each j ,

$$\sum \{\lg(\text{occ}(a_j, S[1..i]) - \lceil \sigma \lg n \rceil) : i \in I_j\} = \lg(n_j - \lceil \sigma \lg n \rceil)! = \lg n_j! - O(\sigma \log^2 n).$$

Therefore, the total number of bits in the encoding is at most

$$\begin{aligned} & \lg n! - \sum_{j=1}^{\sigma} \lg n_j! + n + O\left(\frac{n}{\log n} + \sigma^2 \log^2 n\right) \\ &= \lg \binom{n}{n_1, \dots, n_{\sigma}} + n + O\left(\frac{n}{\log n} + \sigma^2 \log^2 n\right) \\ &\leq n(H+1) + O\left(\frac{n}{\log n} + \sigma^2 \log^2 n\right). \end{aligned}$$

References



- 1 Newton Faller. An adaptive system for data compression. In *Record of the 7th Asilomar Conference on Circuits, Systems and Computers*, pages 593–597, 1973.
- 2 Michael L Fredman and Dan E Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993.
- 3 Travis Gagie. Dynamic length-restricted coding. Master’s thesis, University of Toronto, 2003.
- 4 Travis Gagie. Dynamic Shannon coding. In *Proceedings of the 12th European Symposium on Algorithms (ESA)*, pages 359–370, 2004.
- 5 Travis Gagie and Yakov Nekrich. Worst-case optimal adaptive prefix coding. In *Proceedings of the 11th Symposium on Algorithms and Data Structures (WADS)*, pages 315–326, 2009.
- 6 Robert Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, 24(6):668–674, 1978.
- 7 David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
- 8 Marek Karpinski and Yakov Nekrich. A fast algorithm for adaptive prefix coding. In *Proceedings of the International Symposium on Information Theory (ISIT)*, pages 592–596, 2006.
- 9 Marek Karpinski and Yakov Nekrich. A fast algorithm for adaptive prefix coding. *Algorithmica*, 55(1):29–41, 2009.
- 10 Gyula O H Katona and Tibor O H Nemetz. Huffman codes and self-information. *IEEE Transactions on Information Theory*, 22(3):337–340, 1976.
- 11 Donald E Knuth. Dynamic Huffman coding. *Journal of Algorithms*, 6(2):163–180, 1985.
- 12 Ruy Luiz Milidiú, Eduardo Sany Laber, and Artur Alves Pessoa. Bounding the compression loss of the FGK algorithm. *Journal of Algorithms*, 32(2):195–211, 1999.
- 13 Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.
- 14 Jan van Leeuwen. On the construction of Huffman trees. In *Proceedings of the 3rd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 382–410, 1976.
- 15 Jeffrey Scott Vitter. Design and analysis of dynamic Huffman coding. In *Proceedings of the 26th Symposium on Foundations of Computer Science (FOCS)*, pages 293–302, 1985.
- 16 Jeffrey Scott Vitter. Design and analysis of dynamic Huffman codes. *Journal of the ACM*, 34(4):825–845, 1987.
- 17 Jeffrey Scott Vitter. Algorithm 673: dynamic Huffman coding. *ACM Transactions on Mathematical Software*, 15(2):158–167, 1989.



Taming Graphs with No Large Creatures and Skinny Ladders

Jakub Gajarský  
University of Warsaw, Poland



Paloma T. Lima  
IT University of Copenhagen, Denmark

Marcin Pilipczuk  
University of Warsaw, Poland

Uéverton S. Souza  
University of Warsaw, Poland
Fluminense Federal University, Niterói, Brazil

Lars Jaffke  
University of Warsaw, Poland
University of Bergen, Norway

Jana Novotná  
University of Warsaw, Poland

Paweł Rzażewski  
University of Warsaw, Poland
Warsaw University of Technology, Poland

Abstract

We confirm a conjecture of Gartland and Lokshtanov [arXiv:2007.08761]: if for a hereditary graph class \mathcal{G} there exists a constant k such that no member of \mathcal{G} contains a k -creature as an induced subgraph or a k -skinny-ladder as an induced minor, then there exists a polynomial p such that every $G \in \mathcal{G}$ contains at most $p(|V(G)|)$ minimal separators. By a result of Fomin, Todinca, and Villanger [SIAM J. Comput. 2015] the latter entails the existence of polynomial-time algorithms for MAXIMUM WEIGHT INDEPENDENT SET, FEEDBACK VERTEX SET and many other problems, when restricted to an input graph from \mathcal{G} . Furthermore, as shown by Gartland and Lokshtanov, our result implies a full dichotomy of hereditary graph classes defined by a finite set of forbidden induced subgraphs into tame (admitting a polynomial bound of the number of minimal separators) and feral (containing infinitely many graphs with exponential number of minimal separators).

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Minimal separator, hereditary graph class

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.58

Funding This research is part of projects that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreements 714704 (LJ, JN, MP, PRz, US) and 948057 (JG), and from the Research Council of Norway Grant Agreement 274526 (LJ).



1 Introduction

For a graph G , a set $S \subseteq V(G)$ is a *minimal separator* if there are at least two connected components A, B of $G - S$ with $N(A) = N(B) = S$ (so that S is an inclusion-wise minimal set that separates a vertex of A from a vertex of B). Around the year 2000, Bouchitté and Todinca presented a theory of minimal separators and related objects called *potential maximal cliques* and showed their usefulness for providing efficient algorithms [2]. In particular, the MAXIMUM WEIGHT INDEPENDENT SET problem (given a vertex-weighted graph, find a subset of pairwise nonadjacent vertices of maximum total weight) can be solved in time bounded polynomially in the size and the number of minimal separators in the graph. This result has been generalized by Fomin, Todinca, and Villanger to a large range of problems that can be defined as finding an induced subgraph of constant treewidth with some CMSO₂-expressible property [3]; this includes, for example, LONGEST INDUCED PATH or MAX INDUCED FOREST, which is by complementation equivalent to FEEDBACK VERTEX SET.



© Jakub Gajarský, Lars Jaffke, Paloma T. Lima, Jana Novotná, Marcin Pilipczuk, Paweł Rzażewski, and Uéverton S. Souza;

licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 58; pp. 58:1–58:8



Leibniz International Proceedings in Informatics

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

When do these metaalgorithmic results give efficient algorithms? In other words, which restrictions on graphs guarantee a small number of minimal separators? On one hand, it is easy to see that an n -vertex chordal graph has $\mathcal{O}(n)$ minimal separators. On the other hand, consider the following two negative examples. For $k \geq 3$, the $(k, 1)$ -prism consists of two k -vertex cliques with vertex sets $X = \{x_1, \dots, x_k\}$ and $Y = \{y_1, \dots, y_k\}$ and a perfect matching $\{x_i y_i \mid i \in [k]\}$. It is easy to see that the $(k, 1)$ -prism has $2^k - 2$ minimal separators: any choice of one endpoint of each edge $x_i y_i$ gives a minimal separator, except for the choices X and Y . The $(k, 3)$ -theta consists of k independent edges $\{x_i y_i \mid i \in [k]\}$, a vertex x adjacent to all vertices x_i and a vertex y adjacent to all vertices y_i (the intuition behind the notation is that the graph consists of k paths of length 3, joining x and y). Again, any choice of one endpoint of each edge $x_i y_i$ gives a minimal separator. Thus, both the $(k, 1)$ -prism and the $(k, 3)$ -theta have an exponential (in the number of vertices) number of minimal separators.

In 2019, Milanič and Pivač initiated a systematic study of the question which graph classes admit a small bound on the number of minimal separators in its members [5, 6]. A graph class \mathcal{G} is *tame* if there exists a polynomial $p_{\mathcal{G}}$ such that for every $G \in \mathcal{G}$ the number of minimal separators of G is bounded by $p_{\mathcal{G}}(|V(G)|)$. Clearly, if \mathcal{G} is tame, then MAXIMUM WEIGHT INDEPENDENT SET and all problems captured by the formalism of [3] are solvable in polynomial time when the input graph comes from \mathcal{G} . On the opposite side of the spectrum, \mathcal{G} is *feral* if there exists $c > 1$ such that for infinitely many graphs $G \in \mathcal{G}$ it holds that G has at least $c^{|V(G)|}$ minimal separators. Following the previous examples, the class of chordal graphs is tame while the class of all $(k, 1)$ -prisms and/or all $(k, 3)$ -thetas (over all k) is feral. Milanič and Pivač provided a full tame/feral dichotomy for hereditary graph classes (i.e., closed under vertex deletion) defined by minimal forbidden induced subgraphs on at most 4 vertices [5, 6].

A subsequent work of Abrishami, Chudnovsky, Dibek, Thomassé, Trotignon, and Vusković [1] indicated that the main line of distinction between tame and feral graph classes should lie around the notion of a k -creature. A k -creature in a graph G is a tuple (A, B, X, Y) of pairwise disjoint nonempty vertex sets such that (i) A and B are connected, (ii) A is anti-adjacent to $Y \cup B$ and B is anti-adjacent to $A \cup X$, (iii) every $x \in X$ has a neighbor in A and every $y \in Y$ has a neighbor in B ; (iv) $|X| = |Y| = k$ and X and Y can be enumerated as $X = \{x_1, \dots, x_k\}$, $Y = \{y_1, \dots, y_k\}$ such that $x_i y_j \in E(G)$ if and only if $i = j$. We say that G is k -creature-free if G does not contain a k -creature as an induced subgraph. Similarly as in the examples of the $(k, 1)$ -prism and the $(k, 3)$ -theta, any choice of one endpoint of every edge $x_i y_i$ gives a minimal separator in the subgraph induced by the creature (which, in turn, can be easily lifted to a minimal separator in G). Hence, if G contains a k -creature as an induced subgraph, it contains at least 2^k minimal separators. In fact, the notion of a k -creature is a common generalization of the examples of the $(k, 1)$ -prism and the $(k, 3)$ -theta. Indeed, the $(k, 3)$ -theta contains a k -creature with $A = \{x\}$ and $B = \{y\}$ while the $(k, 1)$ -prism contains a $(k - 2)$ -creature with $A = \{x_{k-1}\}$, $B = \{y_k\}$, $X = \{x_1, \dots, x_{k-2}\}$, and $Y = \{y_1, \dots, y_{k-2}\}$. In particular, Abrishami et al. conjectured that if for a hereditary graph class \mathcal{G} there exists k such that no $G \in \mathcal{G}$ contains a k -creature as an induced subgraph, then \mathcal{G} is tame. (Observe that a presence of arbitrarily large creatures in a hereditary graph class does not immediately imply that the graph class is feral, as the sets A and B can be of superpolynomial size in k .)

A counterexample to the conjecture of [1] has been provided by Gartland and Lokshtanov in the form of a k -twisted ladder [4]. They observed that, despite the fact that the conjecture of [1] is false, every example they can construct “looks like a twisted ladder”, which indicates that the tame/feral boundary for hereditary graph classes should not be far from the said

conjecture. To support this intuition, they introduced the notion of a *k-skinny ladder* (a graph consisting of two induced antiadjacent paths $P = (p_1, \dots, p_k)$, $Q = (q_1, \dots, q_k)$, and independent set $R = (r_1, \dots, r_k)$, and edges $\{p_i r_i, q_i r_i \mid i \in [k]\}$), noted that a *k-skinny-ladder* is an induced minor of every counterexample they constructed, and proved the following.

► **Theorem 1.** *For every k there exists a constant c_k such that if a graph G is k -creature-free and does not contain a k -skinny-ladder as an induced minor, then the number of minimal separators in G is bounded by $c_k |V(G)|^{c_k \log |V(G)|}$, that is, quasi-polynomially in the size of G .*

Gartland and Lokshtanov conjectured that this dependency should be in fact polynomial. Our main result of this paper is a proof of this conjecture.

► **Theorem 2.** *For every $k \in \mathbb{N}$ there exists a polynomial q of degree $\mathcal{O}(k^3 \cdot (8k^2)^{k+2})$ such that every graph G that is k -creature-free and does not contain k -skinny-ladder as an induced minor contains at most $q(|V(G)|)$ minimal separators.*

That is, every hereditary graph class \mathcal{G} for which there exists k such that no member of \mathcal{G} contains a k -creature nor k -skinny-ladder as an induced minor, is tame.

As proven in [4], Theorem 2 implies a dichotomy result into tame and feral graph classes for all hereditary graph classes defined by a finite list of forbidden induced subgraphs. (For the exact definitions of graphs in the statement, we refer to [4].)

► **Theorem 3.** *Let \mathcal{G} be a graph class defined by a finite number of forbidden induced subgraphs. If there exists a natural number k such that \mathcal{G} does not contain all k -theta, k -prism, k -pyramid, k -ladder-theta, k -ladder-prism, k -claw, and k -paw graphs, then \mathcal{G} is tame. Otherwise \mathcal{G} is feral.*

Our proof builds upon the proof of Theorem 1 of [4] and provides a new way of analysing one of the core invariants. For a graph G and a set S , define

$$\zeta_G(S) = \max\{|I| : I \subseteq S \text{ is an independent set and for every } v \notin S \text{ we have } |N(v) \cap I| \leq 1\}.$$

That is, we want a set $I \subseteq S$ of maximum possible size that is not only independent, but no vertex outside S is adjacent to more than one vertex of I . In the proof of Theorem 1 of [4], an important step is to prove that a minimal separator S with huge $\zeta_G(S)$ gives rise to a large skinny ladder as an induced minor. Our main technical contribution is an improved way of analysing minimal separators S with small $\zeta_G(S)$.

► **Theorem 4.** *For every $k, L \in \mathbb{N}$ there exists a polynomial p of degree $\mathcal{O}(k^3 \cdot L)$, such that the following holds. For every k -creature-free graph G , the number of minimal separators S satisfying $\zeta_G(S) \leq L$ is at most $p(|V(G)|)$.*

After brief preliminaries in Section 2, we prove Theorem 4 in Section 3. We show how Theorem 4 implies Theorem 2 (with the help of some tools from [4]) in Section 4.

2 Preliminaries

Let G be a graph, v be a vertex of G , and S be a subset of vertices. By $N_G(v)$ we denote the set of neighbors of v . Similarly, by $N_G(S)$ we denote the set $\bigcup_{x \in S} N_G(x) \setminus S$. If the graph G is clear from the context, we simply write $N(v)$ and $N(S)$.

For sets A, B, C , whenever we write $A \setminus B \setminus C$, the set difference operation associates from the left, meaning that $A \setminus B \setminus C$ is equivalent to $(A \setminus B) \setminus C$ (and, alternatively, to $A \setminus (B \cup C)$).

By $G - S$ we denote the graph obtained from G by deleting all vertices from S along with incident edges, and by $G[S]$ we denote the graph induced by the set S , i.e., $G - (V(G) \setminus S)$. By $\text{CC}(G)$ we denote the set of connected components of G , given as vertex sets.

A *matching* in G is a set of pairwise disjoint edges. We say that a matching $\{x_i y_i \mid i \in [k]\}$ is a *semi-induced matching between* $\{x_1, \dots, x_k\}$ and $\{y_1, \dots, y_k\}$ if for all $i, j \in [k]$, $x_i y_j \in E(G)$ if and only if $i = j$.

For vertices u, v , a set $S \subseteq V(G) \setminus \{u, v\}$ is a *u - v -separator* if u and v are in different connected components of $G - S$. We say that S is a *minimal u - v -separator* if it is a u - v -separator and no proper subset of S is a u - v -separator. A set S is a *minimal separator* if it is a minimal u - v -separator for some u, v . Equivalently, S is a minimal separator if there are at least two components $A, B \in \text{CC}(G - S)$ such that $N(A) = N(B) = S$. Any component $A \in \text{CC}(G - S)$ with $N(A) = S$ is called *full to S* ; a minimal separator has at least two full components.

We define

$$S_G^v = \{N(v) \cap S : v \notin S \text{ and } S \text{ is a minimal separator of } G\}.$$

The following result of Gartland and Lokshtanov will be a crucial tool used in our argument.

► **Lemma 5** (Gartland and Lokshtanov [4]). *If G is a k -creature-free graph, then for every $v \in V(G)$ it holds that $|S_G^v| \leq |V(G)|^{k+1}$.*

Let us also recall the crucial definition. For a set $S \subseteq V(G)$ we define

$$\zeta_G(S) = \max\{|I| : I \subseteq S \text{ is an independent set and for every } v \notin S \text{ we have } |N(v) \cap I| \leq 1\}.$$

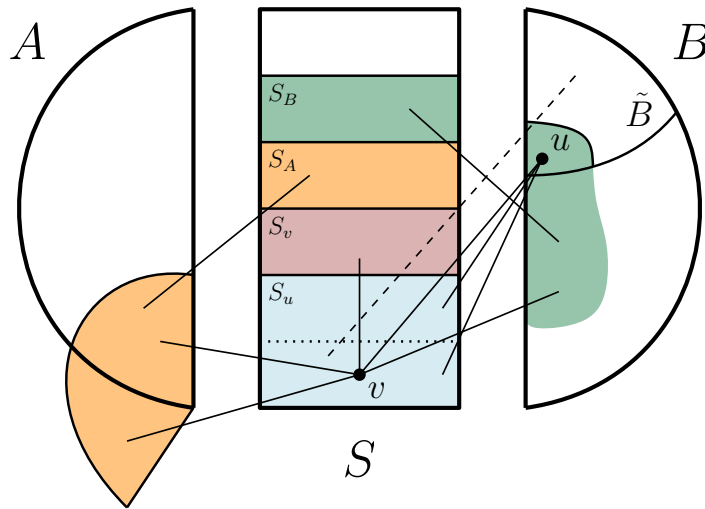
3 Proof of Theorem 4

We prove the theorem by induction on L with the exact bound of $n^{L(4+(k^2+2)(k+2))}$ minimal separators.

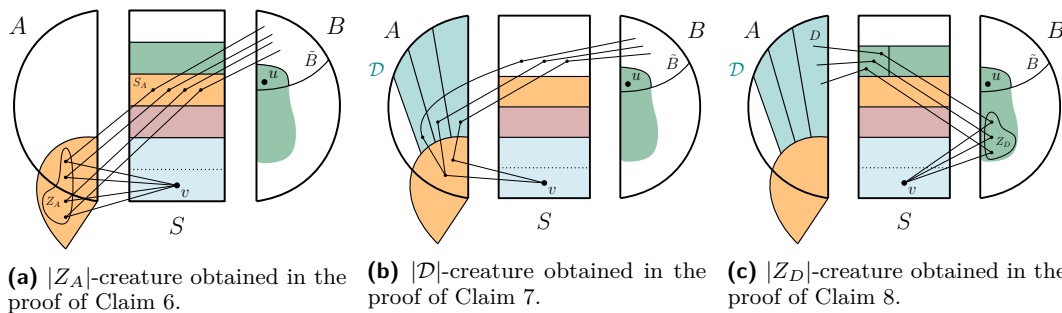
Note that if $S \neq \emptyset$, then $\zeta_G(S) \geq 1$, since for any $u \in S$, the set $I = \{u\}$ satisfies the required properties. Thus, in the base case, when $L = 0$, the only candidate for S is the empty set, therefore the claim holds vacuously. Also, the claim is immediate for $n = 1$, so we assume $n > 1$.

Let S be a minimal separator of G , and let A and B be two connected components of $G - S$ that are full to S . If there is a vertex $v \in V(G) \setminus S$ such that $N(v) \supseteq S$, then $S \in S_G^v$. There are at most n^{k+2} such separators S by Lemma 5; we may therefore assume that no such vertex exists. Let \tilde{B} be a minimal connected subset of B that still dominates S , i.e. such that $N(\tilde{B}) \supseteq S$. Let $u \in \tilde{B}$ be such that $\tilde{B} \setminus \{u\}$ is still connected. Such a vertex u can be found, for instance, as a leaf of a spanning tree of \tilde{B} . We define the following sets that will be important throughout the proof, see Figure 1.

- We let $v \in S \cap N(u) \setminus N(\tilde{B} \setminus \{u\})$. In words, v is a private neighbor (with respect to \tilde{B}) of u in S . Such a vertex v exists by the minimality of \tilde{B} .
- $S_u = N(u) \cap S$.
- $S_v = (N(v) \cap S) \setminus S_u$.
- $S_A = (S \setminus S_u \setminus S_v) \cap N(N(v) \setminus S \setminus B)$. That is, S_A contains the vertices of $S \setminus S_u \setminus S_v$ that have a common neighbor with v in $N(v) \setminus S \setminus B$.
- $S_B = (S \setminus S_u \setminus S_v \setminus S_A) \cap N(N(v) \cap B)$. Similarly, S_B contains the vertices of $S \setminus S_u \setminus S_v \setminus S_A$ that have a common neighbor with v in $N(v) \cap B$.



■ **Figure 1** Subsets of S defined in the proof of Theorem 4. The full lines indicate adjacencies. The dotted line inside S_u indicates a partition of S_u between the private neighbors of u (below) and other neighbors of u (above). The dashed line indicates there is no edge between the sets.

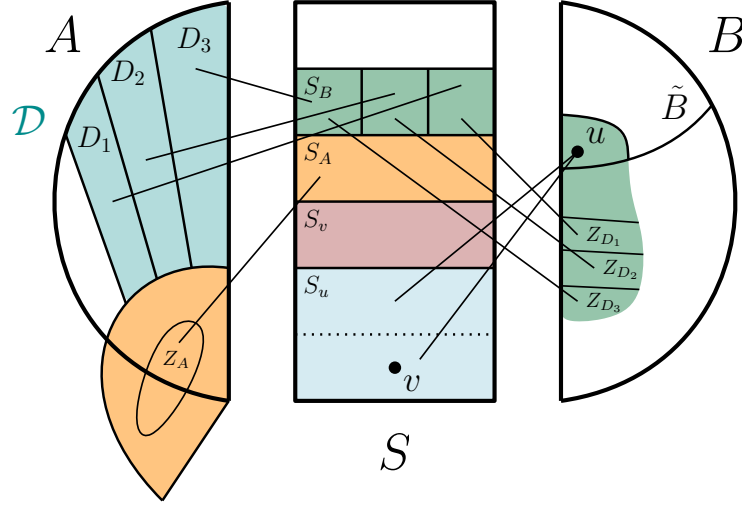


■ **Figure 2** The creatures of Theorem 4.

Our goal is now to identify a small set that dominates $S^* = S_u \cup S_v \cup S_A \cup S_B$. We will repeatedly use Lemma 5 on the vertices of this set in order to bound the number of choices for S^* . We then show that we can find a minimal separator S_0 in $S \setminus S^*$ such that A is a full component in $G - (S^* \cup S_0)$ and there is a component containing $\tilde{B} \setminus \{u\}$. We will be able to show that $\zeta_{G-S^*}(S_0) < \zeta_G(S)$ which allows us to conclude using the induction hypothesis on $G - S^*$.

▷ **Claim 6.** Let $Z_A \subseteq N(v) \setminus S \setminus B$ be a minimal set such that $N(Z_A) \supseteq S_A$. Then G contains a $|Z_A|$ -creature.

Proof. By the minimality of Z_A , each vertex of Z_A has a private neighbor in S_A . Hence, there is a semi-induced matching between Z_A and a size- $|Z_A|$ subset of S_A , say $S_{A,Z}$. We obtain the $|Z_A|$ -creature by considering the sets $(\{v\}, \tilde{B} \setminus \{u\}, Z_A, S_{A,Z})$, see Figure 2a. Indeed, note that by our choice of u , we have that $G[\tilde{B} \setminus \{u\}]$ is connected; v has no neighbors in $\tilde{B} \setminus \{u\}$, as it is a private neighbor of u ; v has no neighbors in $S_{A,Z}$ since $S_{A,Z} \subseteq S_A \subseteq S \setminus S_u \setminus S_v$; clearly there are no edges between $\tilde{B} \setminus \{u\}$ and Z_A since S is a separator; v dominates Z_A and $\tilde{B} \setminus \{u\}$ dominates $S_{A,Z}$. ◁



■ **Figure 3** Illustration of how Q is obtained in the proof of Theorem 4.

▷ **Claim 7.** Let $\mathcal{D} \subseteq \text{CC}(G[A \setminus N(v)])$ be a minimal set of such components that dominates $S \setminus S_u \setminus S_v \setminus S_A$. Then G contains a $|\mathcal{D}|$ -creature.

Proof. By the minimality of \mathcal{D} , for each $D \in \mathcal{D}$ there exists a vertex $y_D \in S \setminus S_u \setminus S_v \setminus S_A$ that is dominated only by vertices of D . Let x_D be a vertex of D that is adjacent to y_D and that is closest to $N(v) \cap A$ in $G[A]$. Note that the edges $\{x_D y_D : D \in \mathcal{D}\}$ form a semi-induced matching between $\{x_D : D \in \mathcal{D}\}$ and $\{y_D : D \in \mathcal{D}\}$ in G . Let P_D be the set of internal vertices on a shortest path between x_D and $N(v) \cap A$ via $G[A]$. Note that $P_D \subset D$ and that P_D is anti-adjacent to $\bigcup_{D \in \mathcal{D}} \{y_D\}$. Indeed, the vertices of P_D are not adjacent to y_D , as this would contradict the minimality of the distance between x_D and $N(v)$; and for any $D' \neq D$, the vertices of P_D are not adjacent to $y_{D'}$ as this vertex is only dominated by vertices of D' , by our choice of $y_{D'}$. Then we obtain a $|\mathcal{D}|$ -creature by considering the sets $(\{v\} \cup (N(v) \cap A) \cup (\bigcup_{D \in \mathcal{D}} P_D), \tilde{B} \setminus \{u\}, \bigcup_{D \in \mathcal{D}} \{x_D\}, \bigcup_{D \in \mathcal{D}} \{y_D\})$, see Figure 2b. Indeed, note that $G[\{v\} \cup (N(v) \cap A) \cup (\bigcup_{D \in \mathcal{D}} P_D)]$ and $G[\tilde{B} \setminus \{u\}]$ are connected; there are no edges between $\{v\} \cup (N(v) \cap A)$ and $\bigcup_{D \in \mathcal{D}} \{y_D\}$ since $(\bigcup_{D \in \mathcal{D}} \{y_D\}) \cap (S_A \cup S_v \cup S_u) = \emptyset$, neither edges between $\bigcup_{D \in \mathcal{D}} P_D$ and $\bigcup_{D \in \mathcal{D}} \{y_D\}$ as mentioned above. Note also that $\{v\} \cup (N(v) \cap A) \cup (\bigcup_{D \in \mathcal{D}} P_D) \cup (\bigcup_{D \in \mathcal{D}} \{x_D\})$ is anti-adjacent to $\tilde{B} \setminus \{u\}$ as argued in the proof of Claim 6. Finally, note that $(N(v) \cap A) \cup (\bigcup_{D \in \mathcal{D}} P_D)$ dominates $\bigcup_{D \in \mathcal{D}} \{x_D\}$, since every x_D either has a neighbor in P_D , or in $N(v) \cap A$ if $P_D = \emptyset$. Finally, it is easy to see that $\tilde{B} \setminus \{u\}$ dominates $\bigcup_{D \in \mathcal{D}} \{y_D\}$. ◁

▷ **Claim 8.** Let \mathcal{D} be as in Claim 7. For each $D \in \mathcal{D}$, let $Z_D \subseteq N(D) \cap B$ be a minimal set that dominates $N(D) \cap S_B$. Then G contains a $|\mathcal{D}|$ -creature.

Proof. By the minimality of Z_D , there is a semi-induced matching between Z_D and a size- $|Z_D|$ subset $S_{B,Z}$ of $N(D) \cap S_B$. We obtain a creature by considering the sets $(\{v\}, D, Z_D, S_{B,Z})$, see Figure 2c. Indeed, note that D is connected by definition; v is not adjacent to $S_{B,Z}$ since $S_{B,Z} \cap (S_u \cup S_v) = \emptyset$ and v is not adjacent to D since D is a connected component of $A \setminus N(v)$; D is not adjacent to Z_D since S is a separator; v dominates Z_D by definition of Z_D and $S_{B,Z} \subseteq N(D)$. ◁

Let $Z = \{u\} \cup Z_A \cup \bigcup_{D \in \mathcal{D}} Z_D$, where Z_A , \mathcal{D} , and Z_D for $D \in \mathcal{D}$ are as defined in Claims 6, 7 and 8, respectively. For all $z \in Z$, let $Q_z = N(z) \cap S$. Let $Q = \bigcup_{z \in Z} Q_z$. Note that Q contains S_u since $u \in Z$, that Q contains S_A since $Z_A \subseteq Z$, and that Q contains S_B . The

latter is due to the fact that the vertices in \mathcal{D} dominate S_B by choice, and each Z_D where $D \in \mathcal{D}$ dominates $N(D) \cap S_B$. We illustrate this situation in Figure 3. It remains to get a grip on S_v .

To do so, let $S' = (S \setminus \{v\}) \cup (N(v) \cap B)$, and note that S' separates $A \cup \{v\}$ from $\tilde{B} \setminus \{u\}$. Let S'' be a minimal subset of S' that still separates $A \cup \{v\}$ from $\tilde{B} \setminus \{u\}$. Note that there are components $A'' \supseteq A \cup \{v\}$ and $B'' \supseteq \tilde{B} \setminus \{u\}$ that are full to S'' and S'' is a minimal separator. Now let $R = N(v) \cap S'' \in S_G^v$, so there are at most n^{k+2} choices for R , by Lemma 5. We observe that $R \supseteq S_v$, which is due to the fact that $\tilde{B} \setminus \{u\}$ dominates S_v , and that S'' separates $\{v\}$ from $\tilde{B} \setminus \{u\}$.

▷ **Claim 9.** There are at most $n^{k^2(k+2)}$ choices for Q , and at most n^{k+2} choices for R .

Proof. We already observed the second statement of the claim above. For the first statement, by Claims 6, 7 and 8, we know that $|Z| < k^2$, so there are at most n^{k^2} choices for Z . For each $z \in Z$, $Q_z \in S_G^z$, so by Lemma 5, there are at most $n^{k^2(k+1)}$ choices for each Q_z , and therefore at most $n^{k^2(k+2)}$ choices for Q . ◁

Now, let $G_0 = G - (Q \cup R)$ and $S_0 = S \setminus Q \setminus R$. Note that $S_0 \subseteq S \setminus S_u \setminus S_v \setminus S_A \setminus S_B$. Moreover, A is a connected component of $G_0 - S_0$, and there is a connected component B_0 of $G_0 - S_0$ that contains $\tilde{B} \setminus \{u\}$. We conclude that S_0 is a minimal separator of G_0 , with A and B_0 being connected components of $G_0 - S_0$ that are full to S_0 . We now show that we can use the induction hypothesis to bound the number of choices for S_0 .

▷ **Claim 10.** $\zeta_{G_0}(S_0) < \zeta_G(S)$.

Proof. Let $I_0 \subseteq S_0$ be an independent set such that for all $y \in V(G_0) \setminus S_0$, $|N_{G_0}(y) \cap I_0| \leq 1$. Let $I = I_0 \cup \{v\}$; I is still an independent set since $S_0 \subseteq S \setminus N_G(v)$. We argue that for all $y \in V(G) \setminus S$, $|N_G(y) \cap I| \leq 1$. Suppose that $y \in N_G(v)$. Since $S_0 \cap (S_A \cup S_B) = \emptyset$, we have that $N_G(y) \cap S_0 = \emptyset$ and therefore $|N_G(y) \cap I| = 1$. We may now assume that $y \notin N_G(v)$. Suppose that $|N_G(y) \cap I| > 1$. Since $y \notin N_G(v)$, we conclude that $y \notin V(G_0) \setminus S_0$, otherwise y would have at least two neighbors in I_0 , a contradiction with the choice of I_0 in S_0 in the graph G_0 . This means that $y \in R \setminus S$, and therefore $y \in N_G(v) \cap B$, which is a contradiction with our assumption that $y \notin N_G(v)$. ◁

The number of choices for u, v, Q , and R is at most $n^{2+(k^2+1)(k+2)}$, see Claim 9. For S_0 , there are at most $n^{(L-1)(4+(k^2+2)(k+2))}$ choices by Claim 10 and the induction hypothesis. Given Q, R , and S_0 , there are at most n choices for $A \in \text{CC}(G - Q - R - S_0)$ and we obtain S as $N(A)$. Taking into account also at most n^{k+2} separators S for which there exists $v \in V(G) \setminus S$ with $S \subseteq N(v)$, the number of separators of G is bounded by

$$\begin{aligned} & n^{2+(k^2+1)(k+2)} \cdot n^{(L-1)(4+(k^2+2)(k+2))} \cdot n + n^{k+2} \\ & \leq n^{4+(k^2+1)(k+2)} \cdot n^{(L-1)(4+(k^2+2)(k+2))} \leq n^{L(4+(k^2+2)(k+2))}. \end{aligned}$$

This completes the proof.

4 Wrapping up the proof of Theorem 2

To conclude the proof of Theorem 2, we observe that the following statement essentially follows from the combinations of Lemma 9 and the proof of Lemma 15 of [4].

► **Lemma 11** ([4]). *If G is a k -creature-free graph that contains a minimal separator S with $\zeta_G(S) > (8k^2)^{k+2}$, then G contains a k -skinny-ladder as an induced minor.*

Proof (sketch). Let G and S be as in the lemma statement. Let $I_0 \subseteq S$ be an independent set of size $\zeta_G(S)$ such that no vertex $v \in V(G) \setminus S$ is adjacent to more than one vertex of I_0 .

Let L_0 and R_0 be two full sides of S . Lemma 9 of [4] asserts that there exists an induced path L in L_0 , an induced path R in R_0 , and a set $I \subseteq I_0$ of size at least $|I_0|/k^2 > (8k^2)^{k+1}$ such that L dominates I and R dominates I .

This is exactly the situation at the end of the first paragraph of the proof of Lemma 15 of [4]. A careful inspection of that proof shows that the remainder of the proof (as well as the invoked Lemmata 8, 13 and 14) do not use other assumptions of Lemma 15. Hence, we obtain the conclusion: a k -skinny ladder as an induced minor of G . ◀

By combining Theorem 4 and Lemma 11, we obtain Theorem 2.

5 Conclusion

In Theorem 2 we showed that if a graph class \mathcal{G} excludes k -creatures as induced subgraphs and k -skinny ladders as induced minors, then \mathcal{G} is tame. However, note that while k -creatures have exponential (in k) number of minimal separators, this is not the case for k -skinny ladders: the class of k -skinny ladders (over all k) is tame. Thus the implication reverse to the one in Theorem 2 does not hold.

Observe that the full tame/feral dichotomy for arbitrary hereditary graph classes is simply false due to some very obscure examples. Let H_k be the $(k, 2^k + 1)$ -theta graph: k paths of length $2^k + 1$ with common endpoints. Note that H_k has $2^{k^2} + 2^{\mathcal{O}(k)}$ minimal separators (2^{k^2} of them choose one internal vertex on each path) and $k2^k + 2$ vertices, so the number of minimal separators of H_k is around $|V(H_k)|^{\log |V(H_k)|}$. Hence, the hereditary class of all induced subgraphs of all graphs H_k for $k \in \mathbb{N}$ is neither tame nor feral.

However, it is still interesting to try to obtain a tighter classification between tame and feral graph classes for some more “well-behaved” hereditary graph classes. As discussed in Conjecture 4 of [4], a good restriction that excludes artificial examples as in the previous paragraph is to focus on induced-minor-closed graph classes.

References

- 1 Tara Abrishami, Maria Chudnovsky, Cemil Dibek, Stéphan Thomassé, Nicolas Trotignon, and Kristina Vušković. Graphs with polynomially many minimal separators. *J. Comb. Theory, Ser. B*, 152:248–280, 2022. doi:10.1016/j.jctb.2021.10.003.
- 2 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232, 2001. doi:10.1137/S0097539799359683.
- 3 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM J. Comput.*, 44(1):54–87, 2015. doi:10.1137/140964801.
- 4 Peter Gartland and Daniel Lokshtanov. Dominated minimal separators are tame (nearly all others are feral). *CoRR*, abs/2007.08761, 2020. arXiv:2007.08761.
- 5 Martin Milanič and Nevena Pivač. Minimal separators in graph classes defined by small forbidden induced subgraphs. In Ignasi Sau and Dimitrios M. Thilikos, editors, *Graph-Theoretic Concepts in Computer Science – 45th International Workshop, WG 2019, Vall de Núria, Spain, June 19-21, 2019, Revised Papers*, volume 11789 of *Lecture Notes in Computer Science*, pages 379–391. Springer, 2019. doi:10.1007/978-3-030-30786-8_29.
- 6 Martin Milanič and Nevena Pivač. Polynomially bounding the number of minimal separators in graphs: Reductions, sufficient conditions, and a dichotomy theorem. *Electron. J. Comb.*, 28(1):P1.41, 2021. URL: <https://www.combinatorics.org/ojs/index.php/eljc/article/view/v28i1p41>, doi:10.37236/9428.

Faster Path Queries in Colored Trees via Sparse Matrix Multiplication and Min-Plus Product

Younan Gao ✉

Faculty of Computer Science, Dalhousie University, Halifax, Canada

Meng He ✉

Faculty of Computer Science, Dalhousie University, Halifax, Canada

Abstract

Let T be an ordinal tree on n nodes in which each node is assigned a color. We consider the batched colored path counting problem and the batched path mode/least frequent element query problem, in which given n query paths, each identified by a pair of nodes in T , one is asked to answer queries of the following forms: How many distinct colors are there on each query path (i.e. the colored path counting problem); what is the color on each query path that occurs at least/most as frequently as any other colors (i.e. the path mode/least frequent element query problem). By reducing the batched colored path counting problem to sparse matrix multiplication, we design a solution that answers n colored path counting queries in $\tilde{O}(n^{\frac{2\omega}{\omega+1}}) = O(n^{1.40704})$ time in total, while we reduce batched path mode/least frequent element query to the min-plus-query-witness problem so that we can answer a batch of n queries in $\tilde{O}(n^{\frac{24+2\omega}{17+\omega}}) = O(n^{1.483814})$ time¹. Previously, both problems could only be solved in $\tilde{O}(n^{1.5})$ time.

Based on similar techniques, we design a dynamic colored path counting structure supporting both queries and updates in $\tilde{O}(n^{\frac{\omega+1}{\omega+3}}) = O(n^{0.627759})$ time, while our dynamic path mode/least frequent element query structures support each operation in $\tilde{O}(n^{\frac{16+\omega(1,2,1)}{26+\omega(1,2,1)}}) = O(n^{0.658139})$ time, where $\omega(1,2,1)$ denotes the minimum value such that the product of an $n \times n^2$ matrix and an $n^2 \times n$ matrix can be computed in $O(n^{\omega(1,2,1)+\epsilon})$ time for any constant $\epsilon > 0$. We also solve batched range mode/least frequent element query problems over arrays in $\tilde{O}(n^{\frac{18+2\omega}{13+\omega}}) = O(n^{1.479603})$ time. Both problems can be viewed as special cases of these batched path queries, and previously, the fastest algorithm for batched range mode queries and batched range least frequent element queries use $O(n^{1.4805})$ and $\tilde{O}(n^{1.5})$ time, respectively.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Information systems → Data structures

Keywords and phrases min-plus product, range mode queries, range least frequent queries, path queries, colored path counting, path mode queries, path least frequent queries

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.59

Funding This work was supported by NSERC of Canada.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

1 Introduction

Trees are used to represent information in many areas of computer science. In tree-structured data, additional properties such as categorical information are often encoded as colors of tree nodes. To facilitate the retrieval of color information, researchers have defined the following queries over an ordinal tree T on n nodes with each node assigned a color from

¹ The \tilde{O} notation hides the $\text{polylog}(n)$ factors, e.g., $O(n \lg^2 n) = \tilde{O}(n)$, and ω denotes the best possible exponent of square matrix multiplication.



$\{0, 1, \dots, C - 1\}$, where $C \leq n$: Given a path in T , a *path colored counting query* returns the number of distinct colors assigned to the nodes in this path, while a *path mode query* or a *path least frequent element query* returns the most frequent or the least frequent color among the multiset of colors assigned to nodes in this path, respectively². The color that is assigned to the most number of nodes in a path is called the *mode* of the path. These queries can be used to compute fundamental statistic information over tree-structured data.

Researchers have studied these query problems and designed data structure solutions [32, 11, 21]. Different time-space tradeoffs have been achieved, and among the best linear-space solutions under word RAM, the structure of He and Kazi [21] can answer a colored path counting query in $O(\sqrt{n} \lg \lg C)$ time, while the solutions of Durocher et al. [11] can answer a path mode query or a path least frequent element query in $O(\sqrt{n/w} \lg \lg n)$ time, where w denotes the number of bits stored in a word. The support for these queries is thus much slower than the support for many other path queries in trees such as path minimum [9, 2, 30, 10, 5, 8], path medium [32, 35, 24, 25], path counting [9, 32, 35, 24, 25] and path majority [11, 13], for which linear-space solutions with sublogarithmic or even constant query times exist.

However, researchers have given evidence to show that these solutions to colored path counting, path mode and path least frequent element are efficient, by proving conditional lower bounds. It has been shown that the multiplication of two $\sqrt{n} \times \sqrt{n}$ Boolean matrices can be performed by answering n colored path counting queries, n path mode queries or n path least frequent element queries. This reduction was explicitly given for colored path counting [21], while for the other two path queries, it follows from the same conditional lower bound on range mode queries [6] and range least frequent element queries [7] in arrays, for which we preprocess an array A , such that, given a range $[i, j]$, we can find the most frequent or least frequent element in $A[i, j]$ efficiently. Note that when the given tree has a single path only, path mode and path least frequent element queries become range mode and range least frequent element queries, respectively. This reduction means, with current knowledge, the total running time of answering n of these path or range queries, including preprocessing, cannot be faster than $n^{\omega/2}$, save for polylogarithmic speedups, where $\omega < 2.37286$ denotes the exponent of matrix multiplication [1]. Furthermore, since the best known combinatorial approach of multiplying two $n \times n$ Boolean matrices require $\Theta(n^3/\text{polylog}(n))$ time [38], the total time of answering n of these queries cannot be faster than $n^{1.5}$, save for polylogarithmic speedups, using pure combinatorial methods with current knowledge. Since the structures of He and Kazi [21] and Durocher et al. [11] can be built in $\tilde{O}(n^{1.5})$ time, they can be used to answer n colored path counting or path mode/least frequent element queries in $\tilde{O}(n^{1.5})$ time, matching this conditional lower bound on pure combinatorial methods within polylogarithmic factors.

The problem of answering n queries given offline is the batched version of these query problems. To achieve $O(n^{1.5-\epsilon})$ -time solutions for some positive constant ϵ , Williams and Xu [37] reduced batched range mode to the min-plus product of a pair of matrices of special structures, which makes it possible to answer a batch of n range mode queries over an array of length n in $O(n^{1.4854})$ time. Gu et al. [17] further improved the running time to $O(n^{1.4805})$. Similar ideas have also yielded dynamic range mode structures with $O(n^{0.655994})$ query and update times [17]. This is surprising as Jin and Xu [27] showed that dynamic range mode

² These problems can also be defined over free trees. However, we follow the definitions given in previous work [11, 21] and assume that T is an ordinal tree, as this allows us to directly apply previous solutions to the problems defined over ordinal trees.

structure cannot simultaneously support update and query in $O(n^{2/3-\epsilon})$ time for any positive constant ϵ using purely combinatorial method with current knowledge. Even before that, Kaplan et al. [31] used sparse matrix multiplication to answer n 2D orthogonal colored range counting queries over n colored points on the plane in $\tilde{O}(n^{\frac{2\omega}{\omega+1}})$ total time. This query counts the number of distinct colors assigned to points in an axis-aligned query rectangle and is related to path colored counting in the sense that they both generalize 1D colored range counting [14, 34] and have the same conditional lower bound.

Despite all these exciting works which beat the conditional lower bounds for combinatorial methods, no previous work was done to solve batched colored path counting or batched path mode in $O(n^{1.5-\epsilon})$ -time. No similar results were achieved for batched range least frequent queries in arrays or batched path least frequent element, either. Therefore, we use matrix multiplication and min-plus product to solve these problems and their dynamic versions.

1.1 Previous Work

The study on colored range counting started in the 1D case, for which Gupta et al. [19] showed a reduction to 2D orthogonal range counting over uncolored points, hence achieving a linear space solution with $O(\lg n / \lg \lg n)$ query time. This problem has been studied extensively in 2D [18, 31, 16, 33, 15], for which Kaplan et al. [29] proved the conditional lower bound based on Boolean matrix multiplication which we discussed previously. They further designed a data structure occupying $O((\frac{n}{t})^2 \lg^6 n + n \lg^4 n)$ words that supports 2D colored range counting in $O(t \lg^7 n)$ time for any $0 < t \leq n$, which was later improved by Gao and He [15] who shaved off several $\log n$ -factors in time/space bounds. Kaplan et al. also showed how to solve batched 2D orthogonal colored range counting in $\tilde{O}(n^{\frac{2\omega}{\omega+1}}) = O(n^{1.40704})$ time by reducing it to sparse matrix multiplication. Recently, Jin and Xu [27] presented a dynamic 2D orthogonal colored range counting structure with $\tilde{O}(n^{2/3})$ query and update times. Colored range counting has also been studied in high dimensions [18, 31, 16]. Finally, He and Kazi [21] considered colored path counting in trees and proved a conditional lower bound which is also based on Boolean matrix multiplication. They designed an $O(n + \frac{n^2}{t^2})$ -word structure that answers queries in $O(t \lg \lg C)$ time for any $t \in [1, n]$, and it can be constructed in $O(\frac{n^2}{t} \lg \lg C)$ time. Hence it implies an $O(n^{3/2} \lg \lg C)$ -time solution to batched colored path counting.

Since Krizanc et al. [32] proposed range and path mode query problems, a long series of papers have been published on these and related problems [32, 6, 7, 12, 37, 36, 17, 27]. The best linear-space solutions include the structure of Chan et al. [6] that answers range mode queries in arrays in $O(\sqrt{n/w})$ time, the structure of Durocher et al. [11] that answers range least frequent element queries in arrays in $O(\sqrt{n/w})$ time, and the structures of Durocher et al. [11] that answer path mode / least frequent elements in trees in $O(\sqrt{n/w} \lg \lg n)$ time. Chan et al. [6] also studied dynamic range mode in arrays, and their solution was later improved by El-Zein et al. [12], whose linear-space structure supports both queries and updates in $O(n^{2/3})$ time. El-Zein et al. also designed a linear-space structure supporting range least frequent element in $O(n^{2/3} \lg n \lg \lg n)$ time and updates in $O(n^{2/3})$ time; different from the original definition of range least frequent, the query is allowed to return a color that does not appear in the query range but appears elsewhere in the array. A Monte Carlo structure is designed in the dynamic case for the original range least frequent element query. It is worth mentioning that all the results summarized in this paragraph use purely combinatorial approaches. They match the conditional lower bounds [6, 7, 27] within polylogarithmic factors.

Recently, more efficient solutions to the batched range mode problem in arrays [36, 37, 17] have been found. Williams and Xu [37] reduced this problem to the min-plus product of a pair of matrices. The second matrix has the property that the entries at each row are non-decreasing, which allows designing a truly subcubic time algorithm for min-plus product of two $n \times n$ matrices. With it, they can solve batched range mode in $O(n^{1.4854})$ time. Later, the query time was improved by Gu et al. [17] to $O(n^{1.4805})$. Sandlund and Xu [36] broke the $O(n^{2/3})$ per-operation time barrier for dynamic range mode in arrays; they reduced the problem to the min-plus-query-witness problem, and achieved a dynamic data structure that supports both queries and updates in $O(n^{0.655994})$ time. Later, Gu et al. [17] further improved the time for each operation to $O(n^{0.6524})$.

1.2 Our Contributions

Our Results. We have achieved the following results:

- an $\tilde{O}(n^{\frac{2\omega}{\omega+1}}) = O(n^{1.40704})$ -time algorithm for batched colored path counting in trees, improving the previous best approach which solves this problem in $\tilde{O}(n^{1.5})$ time [21];
- an $\tilde{O}(n^{\frac{24+2\omega}{17+\omega}}) = O(n^{1.483814})$ -time algorithm for batched path mode/least frequent element queries in trees, improving the previous best result with $\tilde{O}(n^{1.5})$ running time [11];
- an $\tilde{O}(n^{\frac{18+2\omega}{13+\omega}}) = O(n^{1.479603})$ time algorithm for batched range mode/least frequent element queries in arrays, while the previous best results solve batched range mode in $O(n^{1.4805})$ time [17] and batched range least frequent element in $\tilde{O}(n^{1.5})$ time [11];
- a dynamic colored path counting structure for trees supporting queries and updates in $\tilde{O}(n^{\frac{\omega+1}{\omega+3}}) = O(n^{0.627759})$ time, while the best dynamic structure for 2D orthogonal colored range counting, as a related problem, supports each operation in $\tilde{O}(n^{2/3})$ time [27];
- a dynamic data structure for path mode/least frequent element queries in trees supporting queries and updates in $\tilde{O}(n^{\frac{16+\omega(1,2,1)}{26+\omega(1,2,1)}}) = O(n^{0.658139})$ worst-case time. Here, $\omega(1, 2, 1)$ denotes the minimum value such that the product of an $n \times n^2$ matrix and an $n^2 \times n$ matrix can be computed in $O(n^{\omega(1,2,1)+\epsilon})$ time for any constant $\epsilon > 0$. These bounds are close to the $O(n^{0.6524})$ query and update times for dynamic range mode [17] which can be viewed as a special case of dynamic path mode, while the previous best result for dynamic range least frequent element supports queries and updates in $\tilde{O}(n^{2/3})$ time [12].

Overview of Our Approach. To achieve these results, we develop new algorithmic ideas to address the challenges we encounter due to the tree topology. The first challenge is how to apply a divide-and-conquer approach to batched path queries. The solution of Williams and Xu [37] to batched range mode recursively divides the input array into halves, and at each level of recursion, they build data structures to answer queries whose ranges straddle the midpoint. This ensures that the set of possible queries considered share subranges instead of being disjoint, facilitating preprocessing. The solution of Kaplan et al. [31] to batched 2D orthogonal colored range counting is based on a similar idea in 2D. To adapt to tree topology, we apply the centroid decomposition of trees recursively instead. Then, in each component obtained as a result of the decomposition, we preprocess for queries whose paths cross the centroid. This decomposition scheme helps us solve all three batched path queries.

When preprocessing for query paths that contain the centroid in a component, we mark a subset of nodes and attempt to use either sparse matrix multiplication or min-plus product as in previous work. However, more twists to previous approaches are needed. In the solution of Kaplan et al. [31] to batched 2D orthogonal colored counting, the matrices that they need to multiply during preprocessing are already sparse. This is however not the case in

our solution to batched path colored counting. To resolve this, we use the properties of our node marking scheme to carefully reduce the problem of multiplying these matrices to the multiplication of two different but related matrices that are sparse. There is a similar challenge for batched path mode. Previous solutions to batched range mode in [36, 17] reduces the preprocessing for each set of query ranges to the min-plus-query-witness problem over two matrices of which the second matrix is monotone, i.e., entries in the same row are non-decreasing. This allows applying strategies such as dividing each entry by a carefully chosen integer and rounding down the result to decrease the number of different entries in the matrix. In our case, due to the tree topology, the second matrix is not monotone, so this way of applying integer division fails to decrease the number of different entries. To resolve this issue, we design a two-tier scheme to mark nodes for preprocessing, and use the properties of this marking scheme to reduce the weights of entries of the second matrix using different formulas depending on at which tier the corresponding tree nodes are marked. This allows us to decrease the number of different entries to speed up preprocessing.

In range mode queries over arrays, the input elements are split into two categories, i.e., frequent elements and infrequent elements, based on the frequency of each distinct element, and the elements in the different categories are processed in different ways. As mentioned above, the min-plus-query-witness problem was considered in [36, 17] to solve dynamic range mode, and it is used to handle frequent elements. Naturally, a data structure that solves dynamic range mode on arrays can answer static range mode queries as well. By combining an existing solution to the min-plus-query-witness problem shown in [17, Lemma 33] and a static data structure that handles infrequent elements shown in [37, Proof of Theorem 6.1], we design a static data structure for range mode queries on arrays. This simple combination leads to a new static data structure with faster preprocessing time and query time.

In this paper, we describe our solutions to batched colored path counting and batched path mode queries to show the details of these ideas. Due to space constraints, our solutions to other problems such as batched range mode queries in arrays, batched range least frequent element queries in arrays, dynamic path colored counting, dynamic path mode queries, and dynamic path least frequent queries are deferred to the full version of this paper.

2 Preliminaries

This section introduces the notation and the previous results used in this paper.

Notation. Given an ordinal tree T , let $|T|$ denote the number of nodes in T , and let \perp denote its root. For any two nodes $x, y \in T$, we use $P_{x,y}$ to represent the path whose endpoints are x and y . Thus, $P_{x,\perp}$ is a root-to-node path. If y is an ancestor of x , then $P'_{x,y}$ is defined to be the path whose endpoints are x and the child of y that is an ancestor of x , i.e., $P'_{x,y} = P_{x,y} \setminus \{y\}$. Furthermore, $c(x)$ denotes the color assigned to x , and $C(P_{x,y})$ (or $C(P'_{x,y})$) denotes the set of colors that appear in $P_{x,y}$ (or $P'_{x,y}$). Finally, T_x refers to the subtree of T rooted at node x , and $\text{parent}(x)$ is the parent of x .

Navigation in colored ordinal trees. Regarding each tree node color as integer label, the input tree, studied in this paper, is both an ordinal tree and a labeled tree. To support the basic navigational operations on it, we apply the succinct representation of ordinal trees by [22] and the result of He et al. [23] on labeled tree representations. The following lemma summarizes the operations used in our solution and the complexity. Following their notation, we call a node (resp. ancestor) assigned color α an α -node (resp. α -ancestor).

► **Lemma 1** ([22, 23]). *Let T be an ordinal tree on n nodes with each node assigned a color from $\{0, 1, \dots, C - 1\}$, where $C \leq n$. A data structure occupying $n \lg C + 2n + o(n \lg C)$ bits can be built over T in $O(n)$ time to support:*

- $\text{depth}_\alpha(x)$ in $O(\lg \lg C)$ time, which returns the number of α -ancestors of node x ,³
- $\text{depth}(x)$ in $O(1)$ time, which returns the number of ancestors of x ;
- $\text{LCA}(x, y)$ in $O(1)$ time, which returns the lowest common ancestor of nodes x and y .

Given a query path $P_{x,y}$ and a color α in a tree T represented by Lemma 1, He and Kazi [21] showed how to use depth_α and LCA to compute the number of appearances of α in $P_{x,y}$ in $O(\lg \lg C)$ time. This implies the support of *colored path emptiness*, which asks whether a color α appears in $P_{x,y}$. We can further use it to compute $\{|C(P_{x,\perp})| : x \in T\}$, i.e., the numbers of distinct colors on all root-to-node paths. To do this, perform a preorder traversal of T , and each time we visit a node x , we compute $|C(P_{x,\perp})|$ as follows: If x is the root, then $|C(P_{x,\perp})|$ is 1. Otherwise, locate $x' = \text{parent}(x)$, and answer a colored path emptiness query to find out whether $c(x)$ appears in $P_{x',\perp}$. If it does, set $|C(P_{x,\perp})| = |C(P_{x',\perp})|$; otherwise, $|C(P_{x,\perp})| = |C(P_{x',\perp})| + 1$. This process uses $O(n \lg \lg C)$ time.

Node sampling. In our solutions, we use the following lemma based on the pigeonhole principle to select a subset of tree nodes and precompute information for them.

► **Lemma 2** ([26]). *Let T be a tree on n nodes whose height is h . Given an integer $t \in [1, n]$, an integer $\ell \in [0, t - 1]$ can be found in $O(n)$ time such that the total number of nodes whose depths are $\ell + i \times t$ for some $i \in [0, \lfloor \frac{h-\ell}{t} \rfloor]$ is at most n/t .*

Sparse rectangular matrix multiplication. We use the following result of Kaplan et al. [31]:

► **Lemma 3** ([31, Theorem 2.5]). *Let A be an $m \times n$ matrix having at most t non-zero entries, where $t \geq m^{\frac{\omega+1}{2}}$. Then, given the list of non-zero entries of A as the input without storing A verbatim, the product of A and the transpose of A can be computed in $O(tm^{\frac{\omega-1}{2}})$ time.*

3 Batched Colored Path Counting

We first design a data structure to answer a restricted version of colored path counting which requires the query path to contain the root (Section 3.1). Then we generalize it to handle arbitrary paths, which yields a new result for batched colored path counting (Section 3.2).

3.1 Color Counting over Paths Containing the Root

Data structures. To design a data structure for a restricted version of colored path counting which requires the query path to contain the root, we first represent the tree T using Lemma 1. As discussed in Section 2, this structure supports colored path emptiness. Then, for an integer parameter $0 < X \leq n$ to be chosen later, we select at most n/X nodes of T using Lemma 2 and mark them. This means we mark nodes at every X levels of T , starting from some level $\ell \in [0, X - 1]$ determined by Lemma 2. In addition, we mark the root node as

³ Note that the structure of He et al. [23] can support $\text{depth}_\alpha(x, i)$ in $O(\lg \frac{\lg C}{\lg w})$ time, faster than what is stated in Lemma 1. However, this requires a string representation with support for rank and select [4] which uses perfect hashing, and it is not known how to construct this structure in $O(n)$ deterministic time. Therefore, we swap it with the string representation of Belazzougui et al. [3] which can be constructed in linear deterministic time and achieve the bounds in Lemma 1.

well. This means the number, m , of nodes that we mark satisfies $m \leq n/X + 1$. We refer to the i -th marked node visited in a preorder traversal as the i -th marked node for short, where i starts from 0, and this node is denoted by x_i . Thus, x_0 is the root. For each marked node x , we precompute $r(x)$ which is the rank of x among marked nodes defined this way, as well as the value $|C(P_{x,\perp})|$. Furthermore, each node in the tree stores a flag indicating whether it is marked, as well as a pointer to the lowest marked proper ancestor.

Next, we construct an $m \times m$ matrix M . For every pair of integers i and j in $[0, m - 1]$, $M[i, j]$ stores $|C(P_{x_i,\perp}) \cap C(P_{x_j,\perp})|$, i.e., the number of distinct colors that appear in both the path between the i -marked node and the root and the path between j -th marked node and the root. It is worth mentioning that our query algorithm to be described later only uses entries of M that correspond to two marked nodes whose lowest common ancestor is the root. The other entries are never used, but we precompute them regardless.

Overall, our data structures use $O(n + (\frac{n}{X})^2)$ words.

Query algorithm. To describe the query algorithm, let $P_{x,y}$ denote a query path containing the root. Since $\perp \in P_{x,y}$, we always have $\text{LCA}(x, y) = \perp$. W.l.o.g., we assume that neither x nor y is the root node. Let x' and y' denote the lowest marked ancestors of x and y , respectively, and we divide the query path $P_{x,y}$ into three disjoint subpaths: $P'_{x,x'}$, $P_{x',y'}$ and $P'_{y,y'}$. Following the inclusion-exclusion principle, we have that $|C(P_{x',y'})| = |C(P_{x',\perp})| + |C(P_{\perp,y'})| - M[r(x'), r(y')]$. Since the three terms at the right-hand side of this formula have all been precomputed, $|C(P_{x',y'})|$ can be computed in constant time. Next, we count the number of distinct colors that appear in $P'_{x,x'}$ but not in $P_{x',y'}$. This can be done by iterating through each node z in $P'_{x,x'}$ in the direction towards x and check whether $c(z)$ appears in $P_{\text{parent}(z),y'}$ by performing a path emptiness query. The number of distinct colors that are in $P'_{y,y'}$ but not in $P_{x,y'}$ can be counted in a similar way. Adding these two counts to $|C(P_{x',y'})|$ yields the answer. Since x (resp. y) and x' (resp. y') are at most $X - 1$ levels apart, the query time is $O(X \lg \lg C)$. This query algorithm is adapted from an algorithm of He and Kazi [21] for arbitrary query paths, though we use a different matrix.

Preprocessing. Lemmas 1 and 2, together with the discussions on how to compute $\{|C(P_{x,\perp})| : x \in T\}$ in Section 2, can be applied to construct all our data structure components except the matrix M in $O(n \lg \lg C)$ time. To compute M , one way is to define an $m \times C$ matrix A , in which entry $A[i, \alpha] = 1$ if color $\alpha \in C(P_{x_i,\perp})$, and it is 0 otherwise. Then we compute M as AA^T , where A^T denotes the transpose of A . However, since C can be as large as n , the multiplication of A and A^T can be costly.

Instead, to compute M , we use the computation of two related but different $m \times m$ matrices as stepping stones, and one of these two, which we call \hat{M} , can be computed via sparse rectangular matrix multiplication [31]. Before defining \hat{M} , we introduce more notation. Let a be a marked node and b its lowest marked proper ancestor. We define $\hat{C}(a) = C(P'_{a,b}) \setminus C(P_{b,\perp})$, i.e., the set of colors that appear on the path from a to b (including a but excluding b) but not on the path between and including b and the root. Since there are at most X nodes in $P'_{a,b}$, we have $|\hat{C}(a)| \leq X$.

With these definitions, we can define \hat{M} as an $m \times m$ matrix, in which, for each pair of integers $i, j \in [0, m - 1]$, $\hat{M}[i, j]$ stores $|\hat{C}(x_i) \cap \hat{C}(x_j)|$. To compute $\hat{M}[i, j]$, we define an $m \times C$ matrix \hat{A} , and for each integer $i \in [0, m - 1]$ and each color $\alpha \in [0, C - 1]$, $\hat{A}[i, \alpha]$ is set to be 1 if $\alpha \in \hat{C}(x_i)$ and 0 otherwise. Then $\hat{M} = \hat{A}\hat{A}^T$. Since each row of \hat{A} indicates whether each color appears in the set $\hat{C}(a)$ for some marked node a , each row has at most X non-zero entries. Since \hat{A} has m rows and $m \leq n/X + 1$, overall \hat{A} has at most $n + X \leq 2n$

non-zero entries only. A list of non-zero entries of \hat{A} can be computed in $O(n)$ time; the details are omitted due to page limit. With these non-zero entries as input, we can apply Lemma 3 to compute \hat{M} in $O(n^{(\omega+1)/2}/X^{(\omega-1)/2})$ time for any $X \in [n^{(\omega-1)/(\omega+1)}, n]$.

The other $m \times m$ matrix that is used to help us compute M is called M' . For each pair of integers $i, j \in [0, m-1]$, entry $M'[i, j]$ stores $|\hat{C}(x_i) \cap C(P_{x_j, \perp})|$. To compute M' , we perform a preorder traversal of T . Each time we visit a marked node x_j , we compute the j -th column of M' as follows: If $x_j = \perp$, then $j = 0$ and, for each $i \in [0, m-1]$, we set entry $M'[i, 0]$ to be 0, since $c(\perp) \notin \hat{C}(x_i)$. If x_j is not the root, we locate the lowest marked proper ancestor, y , of x_j . Since y is visited before x_j , $M'[i, r(y)]$ has already been computed, and we set $M'[i, j] = M'[i, r(y)] + \hat{M}[i, j]$. To see the correctness, observe that $M'[i, j] = |\hat{C}(x_i) \cap C(P_{x_j, \perp})| = |\hat{C}(x_i) \cap (C(P_{y, \perp}) \cup \hat{C}(x_j))|$; since $C(P_{y, \perp}) \cap \hat{C}(x_j) = \emptyset$, this is equal to $|\hat{C}(x_i) \cap C(P_{y, \perp})| + |\hat{C}(x_i) \cap \hat{C}(x_j)| = M'[i, r(y)] + \hat{M}[i, j]$. This way we can compute M' in $O(n + (\frac{n}{X})^2)$ time provided that \hat{M} is available.

After computing \hat{M} and M' , we can compute M by performing another preorder traversal of T . Each time we visit a marked node x_i , we compute the i -th row of M as follows: If $x_i = \perp$, then $M[i, j] = 1$ for any $j \in [0, m-1]$. Otherwise, we locate the lowest marked proper ancestor, y , of x_i . Since y is visited before x_i , $M[r(y), j]$ has already been computed, and we set $M[i, j] = M[r(y), j] + M'[i, j]$. To see the correctness, observe that $M[i, j] = |C(P_{x_i, \perp}) \cap C(P_{x_j, \perp})| = |(C(P_{y, \perp}) \cup \hat{C}(x_i)) \cap C(P_{x_j, \perp})|$; since $C(P_{y, \perp}) \cap \hat{C}(x_i) = \emptyset$, this is equal to $|C(P_{y, \perp}) \cap C(P_{x_j, \perp})| + |\hat{C}(x_i) \cap C(P_{x_j, \perp})| = M[r(y), j] + M'[i, j]$. In this way, we can compute M in $O(n + (\frac{n}{X})^2)$ time after computing \hat{M} and M' . The total preprocessing time is hence $O(n \lg C + (\frac{n}{X})^2 + \frac{n^{(\omega+1)/2}}{X^{(\omega-1)/2}})$ for any integer $X \in [n^{\frac{\omega-1}{\omega+1}}, n]$.

3.2 Color Counting on an Arbitrary Path

We now generalize the structure in the previous section to support queries over arbitrary paths. Our strategy is to decompose T recursively using *centroid decomposition* [28]; a *centroid* of an n -node tree is a node whose removal splits the tree into connected components each containing at most $n/2$ nodes, and this node can be found in $O(n)$ time.

At level 0 of the recursion, the given tree T is a connected component by itself and we call it the level-0 component. We find a centroid, u , of T , and define a new rooted tree T^u by designating u as the root of T , reorienting edges when necessary. Then we build the query structure in Section 3.1 over T^u . Afterwards, we remove u from T , and build our data structure recursively over each connected component that has more than X nodes. In general, at the i -th level of the recursion, we have a set of connected components called *level- i components* obtained by removing from T the centroids computed in previous levels of the recursion. For each component, we compute its centroid and designate the centroid as the root of this component to build the query structure of Section 3.1. One minor detail is that, before building the query structure over a component of size n' , we need to ensure that colors are encoded as nonnegative integers less than n' . Thus, when $n' \leq C$, we sort the colors that appear in this component using integer sorting in $O(n' \lg \lg n') = O(n' \lg \lg C)$ time [20] and re-encode these colors using their ranks. Then we remove the centroid of each level- i component to split it into a set of level- $(i+1)$ components and recurse. When a component has at most X nodes, we no longer apply this recursive procedure to it, and we call it a *base component*. Thus, we have $O(\lg(n/X))$ recursion levels.

In addition, for each node $x \in T$, we store a list of $O(\lg(n/X))$ pointers, and the i -th pointer maps x to its copy in a level- i component; this pointer is a null if x is removed as a centroid node found in a previous level. Furthermore, we build a weighted tree T' by assigning weights to the nodes of T as follows: If a node x is chosen as the centroid node of

a level- i component, its weight is i . If x is never chosen as a centroid, then its weight is ∞ . Then we construct the linear-space data structure of Chan et al. [8, Theorem 1.1] to support *path minimum queries* over T' in constant time; a path minimum query returns the node with minimum weight in a given query path.

Since we have $O(\lg(n/X))$ recursion levels, both the space costs and construction time of this new structure is a factor of $O(\lg(n/X))$ more than those of the structure in Section 3.1; the detailed analyses are deferred to the full version of this paper. To answer a query with $P_{x,y}$ as the query path, query T' to find the smallest weight, s , assigned to nodes in $P_{x,y}$. If $s = \infty$, then x and y are in the same base component, and thus $P_{x,y}$ has at most X vertices. We can traverse $P_{x,y}$ in T to count the number of distinct colors; each time we visit a new node, we perform a path emptiness query to determine whether we have already encountered this color. This way we can answer the query in $O(X \lg \lg C)$ time. Otherwise, observe that no nodes in $P_{x,y}$ are chosen as centroids for recursion level $s - 1$ or smaller. Therefore, x and y must reside in a same level- s component S , and $P_{x,y}$ contains the centroid of S . Since this centroid is designated as the root of S before building the query structure of Section 3.1 over S , we can use this query structure to answer the query in $O(X \lg \lg C)$ time. Thus we have:

► **Lemma 4.** *Let T be a colored ordinal tree on n nodes with each node assigned a color from $\{0, 1, \dots, C-1\}$, where $C \leq n$, and let X be an arbitrary integer in $[n^{\frac{\omega-1}{\omega+1}}, n]$. A data structure of $O((\lg \frac{n}{X})(n + \frac{n^2}{X^2}))$ words can be constructed in $O((\lg \frac{n}{X})(n \lg \lg C + (\frac{n}{X})^2 + \frac{n^{(\omega+1)/2}}{X^{(\omega-1)/2}}))$ time to support colored path counting query in $O(X \lg \lg C)$ time.*

We finally solve the batched colored path counting problem by first building the query structure of Lemma 4 and then using it to answer n queries. Setting $X = n^{\frac{\omega-1}{\omega+1}}$ yields:

► **Theorem 5.** *A batch of n colored path counting queries over a colored tree T on n nodes can be answered in $\tilde{O}(n^{\frac{2\omega}{\omega+1}}) = O(n^{1.40704})$ time in total.*

4 Batched Path Mode Queries

To solve batched path mode queries over tree T , let t_1 and t_2 be two constant parameters such that $0 \leq t_2 \leq t_1 \leq 1$. We categorize node colors into two different types: a color α is an *infrequent color* if it is assigned to at most n^{1-t_1} nodes in T ; otherwise, we call it a *frequent color*. Thereby, a mode of a query path could be either a frequent or an infrequent color. We use the following lemma to find the most frequent element in the multiset of infrequent colors assigned to the nodes in a query path $P_{x,y}$; due to space constraints, we defer the proof to the full version of this paper.

► **Lemma 6.** *An $O(n^{2-t_1} \lg^4 n)$ -word structure can be constructed in $O(n^{2-t_1} \lg^5 n)$ time, such that, given a query path $P_{x,y}$, the most frequent element and its frequency in the multiset of infrequent colors assigned to the nodes in $P_{x,y}$ can be computed in $O(n^{1-t_1} \lg^5 n)$ time.*

To find the most frequent element in the multiset of frequent colors that appear in $P_{x,y}$, we mark $O(n^{1-t_2})$ nodes using a two-level marking scheme (Section 4.1). Section 4.2 then handles the case in which the endpoints of a query path containing the root are both marked. Finally, We assemble all components in Section 4.3 and solve batched range mode.

4.1 Marking $O(n^{1-t_2})$ Nodes

Let X be an integer parameter in $[n^{t_2}, n]$ to be determined later. We choose at most n/X nodes of T using Lemma 2, mark these nodes and the root, and call them *tier-1 marked nodes*. Since the tier-1 marked nodes form a subset of levels of T , removing them splits T

into a forest of subtrees. Then we use the same lemma to mark nodes at every $\lceil n^{t_2} \rceil$ levels of each subtree starting from some level of the subtree, and these nodes are called *tier-2 marked nodes*. Since the total number of nodes over all subtrees is less than n , there are $O(n^{1-t_2})$ tier-2 marked nodes. We regard both tier-1 and tier-2 marked nodes as *marked nodes*, and there are $O(n/X + n^{1-t_2}) = O(n^{1-t_2})$ marked nodes in total. It follows that a path that connects any non-root node to its lowest marked proper ancestor contains no more than $\lceil n^{t_2} \rceil + 1$ nodes. As before, we refer to the i -th marked node, x_i , visited in a preorder traversal as the i -th *marked node* for short, starting from 0, and $r(x')$ is the rank of the marked node x' .

4.2 Paths with Marked Endpoints Containing the Root: Frequent Colors

Now, for a path $P_{x',y'}$ such that x' and y' are both marked nodes and $\perp \in P_{x',y'}$, we show how to find the most frequent element and its frequency in the multiset of frequent colors assigned to nodes in $P_{x',y'} \setminus \{\perp\} = P'_{x',\perp} \cup P'_{y',\perp}$. Note that for each color other than $c(\perp)$, its frequencies in $P'_{x',\perp} \cup P'_{y',\perp}$ and in $P_{x',y'}$ are exactly the same, while later we consider $c(\perp)$ separately. A frequent color appears more than n^{1-t_1} times in T , so there are only $O(n^{t_1})$ distinct frequent colors. We number the frequent colors incrementally starting from 0 in an arbitrary order, and we refer to the frequent color numbered by k as color f_k . Let μ denote the total number of marked nodes, and let κ denote the total number of distinct frequent colors. Then $\mu = O(n^{1-t_2})$ and $\kappa = O(n^{t_1})$. Next, we construct a $\mu \times \kappa$ matrix M . Corresponding to marked node x_i and frequent color f_k , $M_{i,k}$ stores the negation of the frequency of f_k in path $P'_{x_i,\perp}$. It follows that the min-plus product⁴ of M and its transpose, denoted by $M \star M^T$, is a $\mu \times \mu$ matrix, in which entry $(M \star M^T)_{i,j}$ stores the negation of the maximum frequency of a frequent color in $P'_{x_i,\perp} \cup P'_{x_j,\perp}$, provided $\perp \in P_{x_i,x_j}$. As before, some entries of this matrix correspond to a pair of nodes whose lowest common ancestor is not the root and are thus never used, but we compute these entries regardless. To compute $M \star M^T$ efficiently, it suffices to solve the following problem using M and M^T as input matrices.

► **Problem 1** (Min-Plus-Query-Witness problem [36]). *Build a data structure upon a pair of input matrices A and B , such that, given two integers i and j and a set S of integers in a query, the index k^* with $A_{i,k^*} + B_{k^*,j} = \min_{k \notin S} \{A_{i,k} + B_{k,j}\}$ can be found efficiently.*

Gu et al. [17] solved this problem with three preprocessing steps, which we generalize for our solution. Henceforth, we use k^* to denote the index such that $M_{i,k^*} + M^T_{k^*,j} = \min_{k \notin S} \{M_{i,k} + M^T_{k,j}\}$. All the proofs omitted from this section will be made available in the full version of this paper.

Preprocessing Step 1. Gu et al. provided an efficient solution to Problem 1 when the second input matrix is monotone, i.e., entries in the same row are non-decreasing. In our case, the second matrix, M^T , does not have such a property due to the tree topology. Therefore, our first step is to generalize their definition of *total range* over a monotone matrix to our notion of *total difference*, defined over an arbitrary $a \times b$ matrix A as $\sum_{j=1}^{b-1} (\sum_{k=0}^{a-1} \llbracket A_{k,j} \neq A_{k,j-1} \rrbracket)$, in which the Iverson bracket $\llbracket A_{k,j} \neq A_{k,j-1} \rrbracket$ evaluates to 1 if $A_{k,j} \neq A_{k,j-1}$ is true and 0 otherwise. We then bound the total difference of M^T :

⁴ Given an $m \times n$ matrix A and an $n \times p$ matrix B , the min-plus product of A and B , denoted by $A \star B$, is the $m \times p$ matrix in which entry $(A \star B)_{i,j} = \min_k \{A_{i,k} + B_{k,j}\}$.

► **Lemma 7.** *The total difference of the matrix M^T is $O(n)$.*

The total difference of M^T is however not small enough to lead to an efficient solution to Problem 1 directly. The strategy of Gu et al. is to divide each entry of the second input matrix by a carefully chosen integer and round down the result, to decrease the total difference of the matrix. However, the same method will fail to decrease the total difference of M^T , as M^T is not monotone, so we design a new approach to construct a matrix \tilde{B} from M^T that has a smaller total difference. To introduce this approach, we define $\hat{\text{parent}}_1(v)$ to be node v 's lowest proper ancestor that is tier-1 marked and $\hat{\text{parent}}(v)$ to be v 's lowest proper ancestor that is either tier-1 or tier-2 marked. For a tier-1 marked node \hat{v}_1 , let $R_k(\hat{v}_1)$ denote the frequency of the frequent color f_k in $P'_{\hat{v}_1, \perp}$, and for a tier-2 marked node \hat{v}_2 , let $R'_k(\hat{v}_2)$ denote the frequency of f_k in $P'_{\hat{v}_2, \hat{\text{parent}}(\hat{v}_2)}$ and $\Psi(\hat{v}_2)$ denote the set of tier-2 marked nodes in $P'_{\hat{v}_2, \hat{\text{parent}}_1(\hat{v}_2)}$. We define parameter W to be $\lfloor n^\theta \rfloor$, where θ is a constant with $0 \leq \theta \leq 1$. With the notations above, we introduce matrix \tilde{B} which has the same size as M^T . For each pair (k, j) , if the j -th column of M^T corresponds to a tier-1 marked node \hat{v}_1 , then $\tilde{B}_{k,j}$ stores $-\lfloor \frac{R_k(\hat{v}_1)}{W} \rfloor$. Otherwise, this column must correspond to a tier-2 marked node \hat{v}_2 ; let \hat{v}_1 denote $\hat{\text{parent}}_1(\hat{v}_2)$ and we set $\tilde{B}_{k,j} = -\lfloor \frac{R_k(\hat{v}_1)}{W} \rfloor - \sum_{u \in \Psi(\hat{v}_2)} \lfloor \frac{R'_k(u)}{W} \rfloor$. Then we can bound the total difference of \tilde{B} :

► **Lemma 8.** *The total difference of \tilde{B} is $O(\frac{n}{X} \cdot n^{t_1} + \frac{n}{W})$. Setting X to be $\lfloor Wn^{t_1} \rfloor$, the total difference of \tilde{B} is $O(n/W)$.*

We define a matrix \tilde{A} , in which entry $\tilde{A}_{i,k} = \lfloor \frac{M_{i,k}}{W} \rfloor$, and a matrix \tilde{C}' of the same size as matrix $M \star M^T$, in which $\tilde{C}'_{i,j}$ stores the $(|S|+1)$ -st smallest element in $\{\tilde{A}_{i,k} + \tilde{B}_{k,j} : k \in [\kappa]\}$. Let $[n]$ denote $\{0, 1, \dots, n-1\}$. For each $i \in [\mu]$ and each $j \in [\mu]$, we define set $L_{i,j}$ to be $\{(\tilde{A}_{i,k} + \tilde{B}_{k,j}, k) : k \in [\kappa]\}$. The small total difference of \tilde{B} allows us to borrow ideas from [17] to design a fast algorithm to compute \tilde{C}' and to build a structure that maintains $L_{i,j}$:

► **Lemma 9.** *Matrix \tilde{C}' can be computed in $\tilde{O}(n^{2-2t_2} + n^{(2-t_2)}/W)$ time using matrices \tilde{A} and \tilde{B} . Furthermore, in the same amount of time, a data structure of $\tilde{O}(n^{2-2t_2} + n^{(2-t_2)}/W)$ words can be constructed upon \tilde{A} and \tilde{B} , such that, given a pair (i, j) and an integer $t \geq 1$, the t smallest pairs in $L_{i,j}$, keyed by the first item of each pair, can be listed in $\tilde{O}(t)$ time.*

Preprocessing Step 2. In this step, we take matrices M , M^T and \tilde{C}' as input and partially solve Problem 1 under certain conditions. Let $\rho \geq 0$ be a parameter to be chosen later; let c be any constant that is no less than 1. For each $r \in [(c+2)\mu^\rho \ln n]$, we construct matrices A^r and B^r as follows: Sample j^r uniformly at random from $[\mu]$. Set $A^r_{i,k}$ to be $M_{i,k} + M^T_{k,j^r} - W \cdot \tilde{C}'_{i,j^r}$ if $|M_{i,k} + M^T_{k,j^r} - W \cdot \tilde{C}'_{i,j^r}| \leq 2(W-1) \cdot (3 + W \cdot n^{(t_1-t_2)})$ and $A^r_{i,k} = \infty$ for all $r' < r$; otherwise, set $A^r_{i,k} = \infty$. In addition, set entry $B^r_{k,j}$ to be $M^T_{k,j} - M^T_{k,j^r}$ if $M^T_{k,j^r} \neq \infty$; otherwise, set $B^r_{k,j} = 0$. Following [17], for a pair (i, k) , if $A^r_{i,k} \neq \infty$ for some r , we call (i, k) *covered*; otherwise it is *uncovered*. We call a triple (i, k, j) *weakly relevant* if $|M_{i,k} + M^T_{k,j} - W \cdot \tilde{C}'_{i,j}| \leq 2(W-1) \cdot (3 + W \cdot n^{(t_1-t_2)})$, and we call a triple (i, k, j) *almost relevant* if $0 \leq \tilde{A}_{i,k} + \tilde{B}_{k,j} - \tilde{C}'_{i,j} \leq \frac{(W-1) \cdot (3 + W \cdot n^{(t_1-t_2)})}{W}$. Let $\omega(1, t_1/(1-t_2), 1)$ denote the smallest number such that the product of an $n \times \lceil n^{t_1/(1-t_2)} \rceil$ matrix and an $\lceil n^{t_1/(1-t_2)} \rceil \times n$ matrix can be computed in $O(n^{\omega(1, t_1/(1-t_2), 1) + \epsilon})$ time for any constant $\epsilon > 0$. We have:

► **Lemma 10.** *Given matrices A^r and B^r , a data structure of $\tilde{O}(2(W-1)(3 + Wn^{(t_1-t_2)}) \cdot n^{(1-t_2) \cdot (\rho + 2 + \frac{t_1}{1-t_2} - \sigma)} + n^{(1-t_2) \cdot (\rho + 1 + \frac{2t_1}{1-t_2} - \sigma)})$ words can be built in $\tilde{O}(2(W-1)(3 + Wn^{(t_1-t_2)}) \cdot n^{(1-t_2) \cdot (\rho + \omega(1, t_1/(1-t_2), 1) + \frac{t_1}{1-t_2} - \sigma)})$ time to partially solve the min-plus-query-witness query*

problem upon matrices M and M^T . More precisely, given a query (S, i, j) , if (i, k^*) has been covered, then k^* can be found in $\tilde{O}(|S| + n^{(1-t_2) \cdot (\rho + \sigma)})$ time, where ρ and σ are two constant parameters with $\rho \geq 0$, $0 \leq \sigma \leq \frac{t_1}{1-t_2}$. The randomized part in this preprocessing step can be derandomized. Furthermore, after preprocessing, the number of triples that are almost relevant and uncovered is $O(n^{(1-t_2) \cdot (2 + \frac{t_1}{1-t_2} - \rho)})$.

Preprocessing Step 3. Finally, for each pair (i, j) , we define $V_{i,j} = \{(M_{i,k} + M_{k,j}^T, k) : (i, k, j) \text{ is an uncovered and almost relevant triple}\}$ and apply the same method in [17]:

► **Lemma 11.** *In $\tilde{O}(n^{2-2t_2} + n^{(2-t_2)}/W + n^{(1-t_2) \cdot (2 + \frac{t_1}{1-t_2} - \rho)})$ time, one can find all almost relevant and uncovered triples and build a data structure of $\tilde{O}(n^{2-2t_2} + n^{(1-t_2) \cdot (2 + \frac{t_1}{1-t_2} - \rho)})$ words upon them, such that, given a pair (i, j) and an integer $t \geq 1$, the t smallest elements in $V_{i,j}$, keyed by the first item in each pair, can be listed in $\tilde{O}(t)$ time.*

The Querying Procedure. The query algorithm is similar to the one shown in [17]. Let (S, i, j) be the query parameters. If (i, k^*) is covered, then we use Lemma 10 to find k^* in $\tilde{O}(|S| + n^{(1-t_2) \cdot (\rho + \sigma)})$ time. Otherwise, we claim that $\tilde{A}_{i,k^*} + \tilde{B}_{k^*,j} - \tilde{C}'_{i,j} \leq \frac{(W-1)(Wn^{t_1-t_2}+3)}{W}$ (the proof will be made available in the full version of this paper); therefore, either (i, k^*, j) is almost relevant, or $\tilde{A}_{i,k^*} + \tilde{B}_{k^*,j} - \tilde{C}'_{i,j} < 0$. If (i, k^*, j) is almost relevant, then $(M_{i,k^*} + M_{k^*,j}^T, k^*)$ is among the $(|S| + 1)$ smallest elements in $V_{i,j}$; thereby, we can find k^* in $\tilde{O}(|S|)$ time using Lemma 11. If $\tilde{A}_{i,k^*} + \tilde{B}_{k^*,j} - \tilde{C}'_{i,j} < 0$, then $(\tilde{A}_{i,k^*} + \tilde{B}_{k^*,j}, k^*)$ is among the $(|S| + 1)$ smallest elements in $L_{i,j}$; we can find k^* for this case in $\tilde{O}(|S|)$ time using Lemma 9. As a result, we have solved the min-plus-query-witness problem over M and M^T and achieved Lemma 12. Recall that parameter W was set to be $\lfloor n^\theta \rfloor$.

► **Lemma 12.** *A data structure of $\tilde{O}(n^{2-2t_2} + n^{(1-t_2)(1-\theta)+1} + n^{(1-t_2)(2 + \frac{t_1}{1-t_2} - \rho)} + n^{(1-t_2)(\rho+1 + \frac{2t_1}{1-t_2} - \sigma)} + n^{2\theta(1-t_2) + (t_1-t_2) + (1-t_2)(\rho+2 + \frac{t_1}{1-t_2} - \sigma)})$ words can be built upon M and M^T in $\tilde{O}(n^{1+(1-t_2)(1-\theta)} + n^{2\theta(1-t_2) + (t_1-t_2) + (1-t_2)(\rho + \omega(1, \frac{t_1}{1-t_2}, 1) + \frac{t_1}{1-t_2} - \sigma)} + n^{(1-t_2)(2 + \frac{t_1}{1-t_2} - \rho)})$ time, such that a query defined in Problem 1 can be answered in $\tilde{O}(|S| + n^{(1-t_2)(\sigma + \rho)})$ time, where $\rho \geq 0$, $0 \leq \sigma \leq \frac{t_1}{1-t_2}$, and $0 \leq \theta \leq 1$.*

As a result, we can apply Lemma 12 to find the most frequent element and its frequency in the multiset of frequent colors assigned to nodes in $P'_{x',\perp} \cup P'_{y',\perp}$ provided that x' and y' are marked and $\perp \in P_{x',y'}$. In the static case, the excluded set S is always set to be empty.

4.3 Mode Queries on an Arbitrary Path

We first represent tree T using Lemma 1. As discussed in Section 2, this structure supports finding the number of appearances of a color on a path in $O(\lg \lg C)$ time. Then we mark $O(n^{1-t_2})$ nodes of T as discussed in Section 4.1. We compute the number of appearances of each color on T to determine whether it is frequent or infrequent. Then we construct the data structures of Lemmas 6 and 12 for queries over infrequent and frequent colors, respectively.

Let $P_{x,y}$ be a query path containing the root. W.l.o.g., we assume that neither x nor y is the root node. If $P_{x,y}$ contains less than two marked nodes, then, by our node-marking strategy, $|P_{x,y}| = O(n^{t_2})$. In this case, a mode on $P_{x,y}$ can be found in $O(n^{t_2})$ time. Otherwise, let x' and y' denote the lowest marked ancestors of x and y , respectively, and we divide $P_{x,y}$ into four disjoint parts: $P'_{x,x'}$, $P'_{x',\perp} \cup P'_{y',\perp}$, \perp and $P'_{y,y'}$. Since there are $O(n^{t_2})$ nodes in $P'_{x,x'} \cup P'_{y,y'} \cup \perp$, it requires $O(n^{t_2} \lg \lg C)$ time to scan the nodes in $P'_{x,x'} \cup P'_{y,y'} \cup \perp$, and for each color encountered, count its occurrences in $P_{x,y}$. Let c_1 be the color with maximum

number of occurrences found this way. Then we apply Lemma 6 to find the infrequent color, c_2 , with maximum frequency in $P_{x',y'}$ in $\tilde{O}(n^{1-t_1})$ time, and we query over the data structure of Lemma 12, setting the excluded set $S = \emptyset$, to find the frequent color, c_3 , with maximum frequency in $P'_{x',\perp} \cup P'_{y',\perp}$ in $\tilde{O}(n^{(1-t_2)(\sigma+\rho)})$ time. We also obtain the frequency of c_2 in $P_{x',y'}$ and the frequency of c_3 in $P'_{x',\perp} \cup P'_{y',\perp}$ when finding c_2 and c_3 . Note that, if the mode is not c_1 , then the mode does not appear in $P'_{x,x'} \cup P'_{y,y'} \cup \perp$, so it must be either c_2 or c_3 . Hence it suffices to compare the frequency of c_1 in $P_{x,y}$, the frequency of c_2 in $P_{x',y'}$, and the frequency of c_3 in $P'_{x',\perp} \cup P'_{y',\perp}$ to find the answer to the query.

Finally, we apply the technique in Section 3.2 to compute the mode in an arbitrary path:

► **Lemma 13.** *Let T be a colored ordinal tree on n nodes with each node assigned a color from $\{0, 1, \dots, C-1\}$, where $C \leq n$. A data structure of $\tilde{O}(n^{2-t_1} + n^{2-2t_2} + n^{(1-t_2)(1-\theta)+1} + n^{(1-t_2)(\rho+1+\frac{2t_1}{1-t_2}-\sigma)} + n^{2\theta(1-t_2)+(t_1-t_2)+(1-t_2)(\rho+2+\frac{t_1}{1-t_2}-\sigma)} + n^{(1-t_2)(2+\frac{t_1}{1-t_2}-\rho)})$ words can be constructed in $\tilde{O}(n^{2-t_1} + n^{2\theta(1-t_2)+(t_1-t_2)+(1-t_2)(\rho+\omega(1,\frac{t_1}{1-t_2},1)+\frac{t_1}{1-t_2}-\sigma)} + n^{(1-t_2)(2+\frac{t_1}{1-t_2}-\rho)} + n^{1+(1-t_2)(1-\theta)})$ time, such that a path mode query can be answered in $\tilde{O}(n^{t_2} + n^{1-t_1} + n^{(1-t_2)(\sigma+\rho)})$ time, where $\rho \geq 0$, $0 \leq \sigma \leq \frac{t_1}{1-t_2}$, and $0 \leq \theta \leq 1$.*

Applying Lemma 13 to batched path mode and setting $t_1 = \frac{10}{17+\omega(1,1,1)}$, where $\omega(1,1,1) \in [2, 2.37286)$, $t_2 = 1 - t_1$, $\theta = \frac{2t_1-1}{t_1}$, $\rho = \frac{4t_1-2}{t_1}$, and $\sigma = \frac{3-5t_1}{t_1}$ yields:

► **Theorem 14.** *A batch of n path mode queries over a colored tree T on n nodes can be answered in $\tilde{O}(n^{\frac{24+2\omega}{17+\omega}}) = O(n^{1.483814})$ time.*

References

- 1 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.
- 2 Stephen Alstrup and Jacob Holm. Improved algorithms for finding level ancestors in dynamic trees. In *International Colloquium on Automata, Languages, and Programming*, pages 73–84. Springer, 2000.
- 3 Djamel Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *ACM Transactions on Algorithms (TALG)*, 16(2):1–54, 2020.
- 4 Djamel Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Transactions on Algorithms (TALG)*, 11(4):1–21, 2015.
- 5 Gerth Stølting Brodal, Pooya Davoodi, and S Srinivasa Rao. Path minima queries in dynamic weighted trees. In *Workshop on Algorithms and Data Structures*, pages 290–301. Springer, 2011.
- 6 Timothy M Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55(4):719–741, 2014.
- 7 Timothy M Chan, Stephane Durocher, Matthew Skala, and Bryan T Wilkinson. Linear-space data structures for range minority query in arrays. *Algorithmica*, 4(72):901–913, 2015.
- 8 Timothy M Chan, Meng He, J Ian Munro, and Gelin Zhou. Succinct indices for path minimum, with applications. *Algorithmica*, 78(2):453–491, 2017.
- 9 Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2(1):337–361, 1987.
- 10 Erik D Demaine, Gad M Landau, and Oren Weimann. On cartesian trees and range minimum queries. In *International Colloquium on Automata, Languages, and Programming*, pages 341–353. Springer, 2009.


- 11 Stephane Durocher, Rahul Shah, Matthew Skala, and Sharma V Thankachan. Linear-space data structures for range frequency queries on arrays and trees. *Algorithmica*, 74(1):344–366, 2016.
- 12 Hicham El-Zein, Meng He, J Ian Munro, and Bryce Sandlund. Improved time and space bounds for dynamic range mode. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018.
- 13 Travis Gagie, Meng He, Gonzalo Navarro, and Carlos Ochoa. Tree path majority data structures. *Theoretical Computer Science*, 833:107–119, 2020.
- 14 Travis Gagie, Juha Kärkkäinen, Gonzalo Navarro, and Simon J Puglisi. Colored range queries and document retrieval. *Theoretical Computer Science*, 483:36–50, 2013.
- 15 Younan Gao and Meng He. Space efficient two-dimensional orthogonal colored range counting. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPICs*, pages 46:1–46:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. Pdoi:10.4230/LIPICs.ESA.2021.46.
- 16 Roberto Grossi and Søren Vind. Colored range searching in linear space. In *Scandinavian Workshop on Algorithm Theory*, pages 229–240. Springer, 2014.
- 17 Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhan Xu. Faster monotone min-plus product, range mode, and single source replacement paths. 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 75:1–75:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. Pdoi:10.4230/LIPICs.ICALP.2021.75.
- 18 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.
- 19 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.
- 20 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.
- 21 Meng He and Serikzhan Kazi. Data structures for categorical path counting queries. In *32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 22 Meng He, J Ian Munro, and S Srinivasa Rao. Succinct ordinal trees based on tree covering. In *International Colloquium on Automata, Languages, and Programming*, pages 509–520. Springer, 2007.
- 23 Meng He, J Ian Munro, and Gelin Zhou. A framework for succinct labeled ordinal trees over large alphabets. *Algorithmica*, 70(4):696–717, 2014.
- 24 Meng He, J Ian Munro, and Gelin Zhou. Data structures for path queries. *ACM Transactions on Algorithms (TALG)*, 12(4):1–32, 2016.
- 25 Meng He, J Ian Munro, and Gelin Zhou. Dynamic path queries in linear space. *Algorithmica*, 80(12):3728–3765, 2018.
- 26 David Hutchinson, Anil Maheshwari, and Norbert Zeh. An external memory data structure for shortest path queries. *Discrete Applied Mathematics*, 126(1):55–82, 2003.
- 27 Ce Jin and Yinzhan Xu. Tight dynamic problem lower bounds from generalized bmm and omv. *arXiv preprint arXiv:2202.11250*, 2022.
- 28 Camille Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869. URL: <http://eudml.org/doc/148084>.
- 29 Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM Journal on Computing*, 38(3):982–1011, 2008.

- 30 Haim Kaplan and Nira Shafrir. Path minima in incremental unrooted trees. In *European Symposium on Algorithms*, pages 565–576. Springer, 2008.
- 31 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.
- 32 Danny Krizanc, Pat Morin, and Michiel Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 12(1):1–17, 2005.
- 33 J Ian Munro, Yakov Nekrich, and Sharma V Thankachan. Range counting with distinct constraints. In *CCCG*, 2015.
- 34 Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Transactions on Database Systems (TODS)*, 39(1):1–21, 2014.
- 35 Manish Patil, Rahul Shah, and Sharma V Thankachan. Succinct representations of weighted trees supporting path queries. *Journal of Discrete Algorithms*, 17:103–108, 2012.
- 36 Bryce Sandlund and Yinzhan Xu. Faster dynamic range mode. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- 37 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 12–29. SIAM, 2020.
- 38 Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. *Information and Computation*, 261:240–247, 2018.

Computing the 4-Edge-Connected Components of a Graph: An Experimental Study

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

The notions of edge-cuts and k -edge-connected components are fundamental in graph theory with numerous practical applications. Very recently, the first linear-time algorithms for computing all the 3-edge cuts and the 4-edge-connected components of a graph have been introduced. In this paper we present carefully engineered implementations of these algorithms and evaluate their efficiency in practice, by performing a thorough empirical study using both real-world graphs taken from a variety of application areas, as well as artificial graphs. To the best of our knowledge, this is the first experimental study for these problems, which highlights the merits and weaknesses of each technique. Furthermore, we present an improved algorithm for computing the 4-edge-connected components of an undirected graph in linear time. The new algorithm uses only elementary data structures, and is implementable in the pointer machine model of computation.

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

Keywords and phrases Connectivity Cuts, Edge Connectivity, Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.60

Related Version *arXiv Version*: <https://arxiv.org/abs/2108.08558> [9]

Supplementary Material *Software (Source Code)*: <https://github.com/4eComponentsALGO/files> archived at `swh:1:dir:f82f28576eba10039b91356977e84253d67c73a3`

Funding *Loukas Georgiadis*: 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.

Giuseppe F. Italiano: Partially supported by MIUR, the Italian Ministry for Education, University and Research, under PRIN Project AHeAD (Efficient Algorithms for HARnessing Networked Data).

Evangelos Kosinas: Supported by the project “Dioni: Computing Infrastructure for Big-Data Processing and Analysis”. (MIS No. 5047222) which is implemented under the Action “Reinforcement of the Research and Innovation Infrastructure”, funded by the Operational Programme “Competitiveness, Entrepreneurship and Innovation” (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund).

1 Introduction

Determining or testing the edge connectivity of a graph $G = (V, E)$, as well as computing notions of connected components or subgraphs, is a classical subject in graph theory, motivated by several application areas (see, e.g., [20]), that has been extensively studied since the 1970’s. 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 $|S| = k$. A cut S is *minimal* if no proper subset of S is a cut of



© Loukas Georgiadis, Giuseppe F. Italiano, and Evangelos Kosinas; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 60; pp. 60:1–60:16

Leibniz International Proceedings in Informatics



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

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 [17], 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 . It is known how to compute the $(k-1)$ -edge cuts, $(k-1)$ -vertex cuts, *k -edge-connected components* and *k -vertex-connected components* of a graph in linear time for $k \in \{2, 3\}$ [6, 12, 19, 23, 26]. The case $k = 4$ has also received significant attention [2, 3, 13, 14], but until very recently, none of the previous algorithms achieved linear running time. In particular, Kanevsky and Ramachandran [13] showed how to test whether a graph is 4-vertex-connected in $O(n^2)$ time. Furthermore, Kanevsky et al. [14] 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 [25]. Using the reduction of Galil and Italiano [6] 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 [21] gave an $O(m + k^2n^2)$ -time algorithm to compute the *k -edge-connected components* of a graph G , for any integer k .

Very recently, linear-time algorithms for computing the 4-edge-connected components of an undirected graph were presented in [8, 18]. Specifically, Nadara, Radecki, Smulewicz, and Sokołowski [18] gave two linear-time algorithms, a simple randomized and a more complicated deterministic algorithm. Georgiadis, Italiano, and Kosinas [8] also presented a linear-time deterministic algorithm that requires less machinery but a more detailed analysis. The main part in these algorithms is the computation of the 3-edge cuts of a 3-edge-connected graph G . The algorithms operate on a depth-first search (DFS) tree T of G [23], with start vertex r , and compute three types of 3-edge cuts $C = \{e_1, e_2, e_3\}$, depending on the number of tree edges in C . We refer to a cut C that consists of t tree edges of T as a *type- t cut of G* . The challenging cases are when C is a type-2 or type-3 cut. To handle type-2 cuts in linear time, both NRSS [18] and GIK [8] require the use of the static tree disjoint-set-union (DSU) data structure of Gabow and Tarjan [5], which is quite sophisticated and not amenable to simple implementations. Moreover, the deterministic algorithm of NRSS also employs a linear-time algorithm for computing offline nearest common ancestors [11]. NRSS [18] provided an elegant way to handle type-3 cuts. Specifically, they showed that computing all type-3 cuts can be reduced, in linear time, to computing type-1 and type-2 cuts, by contracting the edges of $G \setminus E(T)$. The algorithm of GIK [8], on the other hand, operates directly on the original graph, but requires a more involved case analysis and, as for type-2 cuts, uses the Gabow-Tarjan DSU data structure.

Our contribution. We present an improved version of the algorithm of GIK [8] for identifying type-2 cuts, so that it only uses simple data structures. The resulting algorithm relies only on basic properties of depth-first search (DFS) [23], and on parameters carefully defined on the structure of a DFS spanning tree (see Section 3.3). As a consequence, it is simple to describe and to implement, and it does not require the power of the RAM model of computation, thus implying the following *new results*:

► **Theorem 1.** *The 3-edge cuts of an undirected graph can be computed in linear time on a pointer machine.*

► **Corollary 2.** *The 4-edge-connected components of an undirected graph can be computed in linear time on a pointer machine.*

Next, we consider the practical performance of these algorithms. We provide carefully engineered implementations of the randomized algorithm of NRSS [18], the deterministic algorithm of GIK [8], as well as our new linear-time pointer-machine algorithm. Then, we conduct a thorough empirical study to highlight the merits and weaknesses of each technique. In particular, our experimental evaluation addresses the following questions:

- How well does the randomized algorithm perform with respect to its deterministic counterparts?
- How efficient is the graph contraction technique in practice?
- How does our new linear-time pointer-machine algorithm compare against the linear-time RAM algorithms?
- How fast can we compute the 4-edge-connected components of a graph compared to computing the k -edge-connected components for $k < 4$?

2 Linear-time algorithms for computing the 4-edge-connected components

Let $G = (V, E)$ be the input graph, which may have multiple edges. To compute the 4-edge-connected components of G , we can perform 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 .

For Steps 1–3, we can compute the 1-edge and 2-edge cuts, and the 2-edge and 3-edge-connected components a graph in linear time by [6, 12, 19, 23, 26]. It can be easily demonstrated that the (subgraphs induced by the) k -edge-connected components of G , for $k = 1, 2$, have the same k' -edge-connected components as G , for all $k' > k$. However, the analogous property does not hold for $k = 3$. Thus we use the construction provided by Dinitz [2], which extends the 3-edge-connected components of G , by adding some extra edges to them, so that the resulting auxiliary graphs have the same k' -edge-connected components as G , for all $k' > 3$. To perform Step 4, we use the fact that we can construct a compact representation of the 2-cuts of any 2-edge-connected component H of G , which allows us to compute its 3-edge-connected components C_1, \dots, C_ℓ in linear time [10, 26]. We let $G[C_i]$ denote the subgraph of G that is induced by the vertices in C_i . By shrinking each 3-edge-connected component C_i of H into a single node, we obtain a quotient graph Q of H which has the structure of a tree of cycles [2], i.e., 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 $G[C_i]$, which acts as a substitute for the cycle of Q that contains (C_i, C_j) and (C_i, C_k) . Now let H_i be the graph $G[C_i]$ plus all those virtual edges. Then H_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 H_i .

Hence, we have reduced the problem of computing the 4-edge-connected components of a (general) graph, to the computation of the 4-edge-connected components of a collection of auxiliary 3-edge-connected graphs.

So, from now on, we let G be a 3-edge-connected graph. We can compute its 4-edge-connected components by successively splitting G 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 refer to [8] for the details.) It remains to describe how to compute the 3-edge cuts of a 3-edge-connected graph G .

3 Computing the 3-cuts of a 3-edge-connected graph

In this section we give an overview of the algorithms of NRSS [18] and GIK [8] for computing the 3-edge cuts of a 3-edge-connected graph G . Then, we also present our new linear-time pointer-machine algorithm. Throughout this section, we assume that $G = (V, E)$ is a 3-edge-connected graph with n vertices and $m \geq 3n/2$ edges, and may have multiple edges. It is well-known that the number of the 3-edge-cuts of G is $O(n)$ [20], which also follows from the definition of the cactus graph [1, 15]). See also [8, 18] for an independent proof of this fact.

For a graph H , we let $V(H)$ and $E(H)$ denote the set of vertices and edges of H , respectively. If H is a subgraph of G then $G \setminus E(H)$ denotes the graph that results from G after deleting the edges of H .

3.1 Depth-first search and related notions

Here we introduce some concepts and parameters that are used in the algorithms, defined with respect to a depth-first search spanning tree. Let T be the spanning tree of G provided by a depth-first search (DFS) of G [23], with start vertex r . A vertex u is an ancestor of a vertex v (v is a descendant of u) in T if the tree path from r to v contains u . Thus, we consider a vertex to be both an ancestor and a descendant of itself. The edges in T are called *tree-edges*; the edges in $E(G) \setminus E(T)$ are called *back-edges*, as their endpoints have ancestor-descendant relation in T . 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). We identify vertices with their preorder number assigned during the DFS. Thus, if v is an ancestor of u in T , then $v \leq u$. Let $T(v)$ denote the set of descendants of v , and let $ND(v) = |T(v)|$ denote the number of descendants of v . Then, vertex u is a descendant of v (i.e., $u \in T(v)$) if and only if $v \leq u < v + ND(v)$ [24].

Whenever (x, y) denotes a back-edge, we shall assume that x is a descendant of y . We say that a back-edge (x, y) leaps over a vertex v if x is a descendant of v and y is a proper ancestor of v . We let $B(v)$, for a vertex $v \neq r$, denote the set of all back-edges that leap over v , and we let $bcount(v) = |B(v)|$ denote the number of elements of $B(v)$. 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 $B(v)$, which implies the following property:

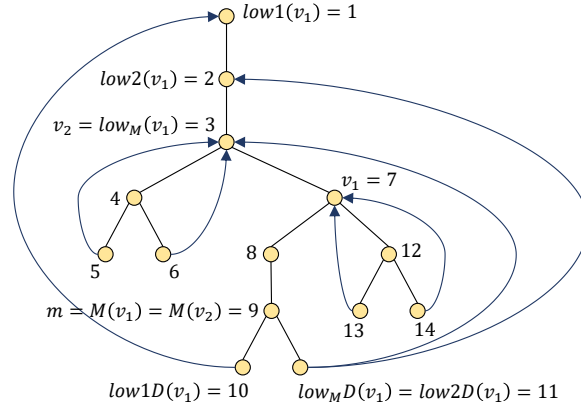
► **Property 3** ([8]). *A connected graph G is 2-edge-connected if and only if $bcount(v) > 0$, for every $v \neq r$. Furthermore, G is 3-edge-connected only if $bcount(v) > 1$, for every $v \neq r$.*

This suggests that the sets $B(v)$ can be used to determine various connectivity relations in graphs. In fact, we can consider many useful notions extracted from those sets by considering the distribution of the ends of the back-edges that are contained in them. First, consider the lower ends of the back-edges in $B(v)$. We define the *low* point of vertex v , denoted by $low(v)$, as the minimum vertex y such that there exists a back-edge $(x, y) \in B(v)$. Formally, $low(v) := \min\{y \mid \exists(x, y) \in B(v)\}$. We also let $lowD(v)$ be x . That is, $lowD(v)$ is a descendant of v from which stems a back-edge that provides $low(v)$. (Notice that $lowD(v)$ is not uniquely determined.) We denote the back-edge $(lowD(v), low(v))$ as $MinUp(v)$. Similarly, we define the *high* point of v , denoted by $high(v)$, as the maximum vertex y such that there exists a back-edge $(x, y) \in B(v)$. Formally, $high(v) := \max\{y \mid \exists(x, y) \in B(v)\}$. We also let $highD(v)$ be x . That is, $highD(v)$ is a descendant of v from which stems a back-edge that provides $high(v)$. (Again, $highD(v)$ is not uniquely determined.) We denote the back-edge $(highD(v), high(v))$ as $MaxUp(v)$. Finally we let $l_1(v)$ be the minimum y such that there exists a back-edge of the form (v, y) (or v , if no such back-edge exists). Formally, $l_1(v) := \min(\{y \mid \exists(v, y) \in B(v)\} \cup \{v\})$. Furthermore, we define $l_2(v) := \min(\{y \mid \exists(v, y) \in B(v) \setminus \{(v, l_1(v))\}\} \cup \{v\})$.

Now we consider the higher ends of the back-edges in $B(v)$. First, we let $MinDn(v)$ (resp. $MaxDn(v)$) denote a back-edge in $B(v)$ with minimum (resp. maximum) higher end. We then define the *maximum* point $M(v)$ of v as the maximum vertex z such that $T(z)$ contains the higher ends of all back-edges in $B(v)$. In other words, $M(v)$ is the nearest common ancestor of all x for which there exists a back-edge $(x, y) \in B(v)$. See Figure 1. (Clearly, $M(v)$ 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 $prevM(v_i) := v_{i-1}$, for every $i \in \{2, \dots, k\}$. Thus, for every vertex v , $nextM(v)$ is the successor of v in the decreasingly sorted list $M^{-1}(M(v))$, and $prevM(v)$ is the predecessor of v in the decreasingly sorted list $M^{-1}(M(v))$.

Now let v be a vertex and let u_1, \dots, u_k be the children of v sorted in non-decreasing order w.r.t. their *low* point. We let $c_i(v)$ be u_i , if $i \in \{1, \dots, k\}$, and \emptyset if $i > k$. (Note that $c_i(v)$ is not uniquely determined, since some children of v may have the same *low* point.) Then we call $c_1(v)$ the *low1* child of v , and $c_2(v)$ the *low2* child of v . 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 $M(v)$. We leave $\tilde{M}(v)$ undefined if no such proper descendant x of $M(v)$ exists. 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 $M(v)$. We leave $M_{low1}(v)$ (resp., $M_{low2}(v)$) undefined if no such proper descendant x of the *low1* (resp., *low2*) child of $M(v)$ exists.

We note that it is easy to compute all $l_1(v)$, $l_2(v)$, $low(v)$, $lowD(v)$, and $bcount(v)$ during the DFS. In particular, the notion of low points plays central role in classic algorithms for computing the biconnected components [23], the triconnected components [12] and the 3-edge-connected components [6, 12, 19, 26] of a graph. (Hopcroft and Tarjan [12] also use a concept of high points, which, however, is different from ours.) Furthermore, all $M(v)$, $\tilde{M}(v)$, $M_{low1}(v)$ and $M_{low2}(v)$, for all vertices v for which they are defined, can be computed in



■ **Figure 1** An illustration of the concepts defined on a depth-first search (DFS) spanning tree of an undirected graph. Vertices are numbered in DFS order and back-edges are shown directed from descendant to ancestor in the DFS tree. Vertices v_1 and v_2 have $M(v_1) = M(v_2) = m$, hence $\{v_1, v_2\} \subseteq M^{-1}(m)$, $nextM(v_1) = v_2$, and $prevM(v_2) = v_1$. Moreover, $(11, 3) \notin B(v_2)$, so $low_M(v_1) = 3$.

linear-time in pointer machines [8]. For the computation of all $high(v)$ (and $highD(v)$), [8] and [18] gave a linear-time algorithm that uses the static tree DSU data structure of Gabow and Tarjan [5], and thus their computation depends on the power of RAM machines.

3.2 Randomized algorithm of NRSS

We give an overview of the randomized linear-time algorithm of Nadara, Radecki, Smulewicz, and Sokołowski [18]. Let $H : E \mapsto 2^E$ be a function defined as follows. If e is a back-edge, $H(e) = \{e\}$. Otherwise, $H(e) = B(u)$, for the unique vertex $u \neq r$ such that $e = (u, p(u))$. Now NRSS provide an elegant characterization of 3-cuts: a triple of edges $\{e_1, e_2, e_3\}$ is a 3-cut if and only if $H(e_1) \oplus H(e_2) \oplus H(e_3) = \emptyset$ (where \oplus denotes the symmetric set difference). Then we get a useful equivalent characterization of 3-cuts by representing the sets $H(e)$ as bit-vectors. To be precise, we consider the composition of $\chi : 2^E \mapsto \{0, 1\}^E$ (the characteristic function on E) with $H : E \mapsto 2^E$; then we have that $\{e_1, e_2, e_3\}$ is a 3-cut if and only if $\chi_{H(e_1)} \oplus \chi_{H(e_2)} \oplus \chi_{H(e_3)} = 0$ (where \oplus here denotes the logical XOR function). Thus, if for an edge e we have a candidate e' that may provide a 3-cut of the form $\{e, e', e''\}$, for a third edge e'' , then e'' is uniquely determined by the expression $\chi_{H(e)} \oplus \chi_{H(e')}$.

Since we cannot compute all $\chi_{H(e)}$, for all edges e , in linear time, the idea of NRSS is to compress the bits of those vectors into a fixed number of RAM words. (We assume that a RAM cell can store $O(\log m)$ bits.) In particular, we select $b = \lceil 3 \lg m \rceil$ random bits to represent the characteristic function of $H(e)$, for every back-edge e , and we call this value $CH(e)$. (We may select a bigger multiple of $\lg m$ for higher precision; however, $\lceil 3 \lg m \rceil$ is enough for all practical purposes.) Then $CH(e)$, for a tree-edge e , corresponds to $\bigoplus_{e' \in H(e)} CH(e')$.

Now, for type-1 and type-2 cuts, we can find good candidate edges that may yield 3-cuts. Specifically, let $\{(u, p(u)), e, e'\}$ be a type-1 cut (i.e., e and e' are back-edges). Then one of e, e' is the back-edge that provides the low point of u . Thus, without loss of generality, we may assume that $e = \text{MinUp}(u)$. Then we can retrieve e' with high probability by determining $CH^{-1}(CH((u, p(u))) \oplus CH(e))$. Now let $\{(u, p(u)), (v, p(v)), e\}$ be a type-2 cut (where e is a back-edge). In this case u and v are related as ancestor and descendant, and thus we may

assume, without loss of generality, that u is a descendant of v . Then we have that either e leaps over u but not over v , or that e leaps over v but not over u . In the first case, e is the back-edge that provides the *high* point of u (that is, $e = \text{MaxUp}(u)$). Thus we can retrieve $(v, p(v))$ with high probability by determining $CH^{-1}(CH((u, p(u))) \oplus CH(\text{MaxUp}(u)))$. In the second case, e is either $\text{MinDn}(v)$ or $\text{MaxDn}(v)$. Thus we can retrieve $(u, p(u))$ with high probability from $CH((v, p(v)) \oplus \text{MinDn}(v))$ or $CH((v, p(v)) \oplus \text{MaxDn}(v))$. NRSS provide linear-time algorithms for computing all $\text{MaxUp}(v)$, $\text{MinDn}(v)$ and $\text{MaxDn}(v)$, for all vertices $v \neq r$, using the static-tree DSU data structure of Gabow and Tarjan [5]. (The inverses CH^{-1} can be computed efficiently by using e.g. hash tables.)

Finally, for type-3 cuts, NRSS provided a simple linear-time reduction to computing type-1 and type-2 cuts. First, compute the connected components C_1, \dots, C_k of $G \setminus E(T)$. Then, form a reduced graph G' by contracting each component C_i into a single vertex c_i . In this construction, we maintain parallel edges between two vertices but remove self-loops. Then, recursively compute the type-1 and type-2 cuts of G' . Assuming that G is 3-edge connected, it is easy to observe that G' satisfies the following properties: (i) G' is also 3-edge connected, and (ii) $|E(G')| \leq \frac{2}{3}|E(G)|$. The latter inequality follows from the fact that the minimum vertex degree in G is 3, and that $|E(G')| \leq |V(G)|$ due to the contractions. Then, the total running time spend during the recursive calls is bounded by $O(m)$.

3.3 Deterministic algorithm of GIK

Here we provide an overview of the deterministic linear-time algorithm of Georgiadis, Italiano, and Kosinas [8]. The idea is to distinguish various types of 3-cuts on a DFS tree of the graph, and then provide an algorithm specifically adapted for each particular case. The classification of GIK is heavily guided by some DFS parameters that can be extracted from the sets $B(v)$ of the back-edges that leap over a vertex v , for $v \neq r$, without computing the sets $B(v)$ explicitly.

First, let $\{(u, p(u)), e, e'\}$ be a type-1 cut. Then we obviously have $B(u) = \{e, e'\}$. Thus, in order to identify all those cuts, we need to have computed, for every vertex $u \neq r$, two back-edges that leap over u . We take as one of them a back-edge e that provides the *low* point of u . To get one more back-edge, we extend the definition of *low* points: we let $\text{low2}(u)$ be the lowest lower end of all back-edges in $B(u) \setminus \{e\}$, and let $\text{low2D}(u)$ be a descendant of u such that $(\text{low2D}(u), \text{low2}(u)) \in B(u) \setminus \{e\}$. Let $\text{low1}(u) := \text{low}(u)$ and $\text{low1D}(u) := \text{lowD}(u)$. (Refer to Figure 1.) Then, for every $u \neq r$, we have that $\{(\text{low1D}(u), \text{low1}(u)), (\text{low2D}(u), \text{low2}(u))\} \subseteq B(u)$, and these two back-edges form a 3-cut with $(u, p(u))$ if and only if $\text{bcount}(u) = 2$. We note that all back-edges $(\text{low2D}(v), \text{low2}(v))$ can be computed easily during the DFS, and thus all type-1 cuts can be computed in linear time in a pointer machine without using sophisticated data structures.

Now let $\{(u, p(u)), (v, p(v)), e\}$ be a type-2 cut (where e is a back-edge). Then we have that u and v are related as ancestor and descendant, and we may assume in what follows, without loss of generality, that u is a descendant of v . Then we have that either $B(u) = B(v) \sqcup \{e\}$ or $B(v) = B(u) \sqcup \{e\}$. In either case, we can exploit the similarity of the sets $B(u)$ and $B(v)$ to find good candidate pairs $\{u, v\}$ that may provide type-2 cuts of the form $\{(u, p(u)), (v, p(v)), e\}$, for a back-edge e . Take the case $B(u) = B(v) \sqcup \{e\}$ as an example. Then e must be the back-edge that provides the *high* point of u , and so $e = (\text{highD}(u), \text{high}(u))$. Depending on the location of $\text{highD}(u)$, we can relate $M(v)$ with a maximum point of u . Specifically, if $\text{highD}(u)$ is a descendant of $M(v)$, then $M(v) = M(u)$, and v is the greatest ancestor of u with this property (i.e., $v = \text{nextM}(u)$). If $\text{highD}(u)$ is an ancestor of $M(v)$, then $M(v) = \tilde{M}(u)$ and v is the greatest ancestor of u with this property. Finally, if $\text{highD}(u)$

is not related as ancestor or descendant with $M(v)$, then $M(v) = M_{low1}(u)$, and v is again the greatest ancestor of u with this property. These considerations are the basis of an efficient algorithm for determining, for a vertex u , a proper ancestor v of u , and a back-edge e , that may yield a type-2 cut of the form $\{(u, p(u)), (v, p(v)), e\}$. (In fact, once such a v has been found, we only have to check whether $bcount(u) = bcount(v) + 1$ to establish the existence of this cut.) The case $B(v) = B(u) \sqcup \{e\}$ can be handled in a similar manner. We refer to [8] for more details.

Finally, with a much more involved subdivision of type-3 cuts into more cases, and with extensive use of DFS parameters extracted from the sets of leaping back-edges, GIK provide linear-time algorithms to identify all those cuts. These algorithms use the *high* points in a way that seems difficult to avoid them. We need not mention any details of those cases because we can avoid them overall using the idea of NRSS for identifying all type-3 cuts on the contracted graph.

In the next section we will show how to handle the cases of type-2 cuts without the use of *high* points, thus providing the first linear-time pointer-machine algorithm for determining all 3-cuts of 3-edge-connected graphs.

3.4 A new linear-time pointer-machine algorithm

Our goal is to provide a method to compute all 3-cuts in linear time without using the *high* points (since the only known algorithm for computing all *high* points in linear time depends on the power of RAM machines [8, 18]). We achieve this through the following three improvements over the GIK algorithm. First, we introduce two new parameters, $low_M D(v)$ and $low_M(v)$, that yield the back-edge $(highD(u), high(u))$ that is needed in the case $B(u) = B(v) \sqcup \{e\}$, $M(u) = M(v)$ (see the previous subsection). Second, we replace all conditions provided by GIK for determining type-2 cuts with equivalent ones that avoid the invocation and the computation of the *high* points. And finally, we use the idea of NRSS to deal with type-3 cuts by reducing them to type-1 and type-2 cuts.

Let v be a vertex such that $nextM(v) \neq \emptyset$. Then we have $B(nextM(v)) \subset B(v)$. Thus we can define $low_M(v)$ as the lowest lower end of all back-edges in $B(v) \setminus B(nextM(v))$, and we let $low_M D(v)$ be a vertex such that $(low_M D(v), low_M(v))$ is a back-edge in $B(v) \setminus B(nextM(v))$. (Refer to Figure 1.) Formally, we have

- $low_M(v) := \min\{y \mid \exists(x, y) \in B(v) \setminus B(nextM(v))\}$.
- $low_M D(v) :=$ a vertex x such that $(x, low_M(v)) \in B(v)$.

All edges $(low_M D(v), low_M(v))$, for all vertices v for which they are defined, can be determined with a linear-time algorithm that uses only elementary data structures and depends on a specific ordering of the adjacency lists of the graph. Now, if we have a pair of vertices $\{u, v\}$ such that $B(u) = B(v) \sqcup \{e\}$, for a back-edge e , and $v = nextM(u)$, then it is certainly true that $e = (low_M D(u), low_M(u)) = (highD(u), high(u))$. This shows how to handle one of the three subcases of the lower type-2 cuts, described in the previous Section. For the other two subcases, as well as for the upper case of type-2 cuts, we can simply provide alternative criteria that characterize them, using most of the DFS notions in Section 3.1 in a way similar to [8] (but without using the *high* points). The algorithms that we get in this way are very similar to those given by GIK.

Due to the space constraints, we refer to [9] for a detailed description of those criteria and their proofs, as well as for an exposition and proof of correctness of the full algorithm for determining type-2 cuts. Throughout the remainder of the paper, we refer to this algorithm as *Linear*.

4 Empirical Analysis

We implemented our algorithms in C++, using g++ 7.5.0 with full optimization (flag -O4) to compile the code. The reported running times were measured on a GNU/Linux machine, with Ubuntu (18.04.6 LTS): a Dell Precision Tower 7820 server 64-bit NUMA machine with an Intel(R) Xeon(R) Gold 5220R processor and 192GB of RAM memory. The processor has 24.75MB of cache memory and 18 cores. In our experiments we did not use any parallelization, and each algorithm ran on a single core. We report CPU times measured with the `high_resolution_clock` function of the standard library `chrono`, averaged over ten different runs.

Implementation details

We tested four implementations, two for the randomized algorithm of Nadara, Radecki, Smulewicz, and Sokołowski [18] (NRSS and NRSS-sort), one for the deterministic algorithm of Georgiadis, Italiano, and Kosinas [8] (GIK), and one for our linear-time pointer machine algorithm (Linear). We did not include an implementation of the deterministic algorithm of [18], because it is more complicated than the routine in GIK for computing type-2 cuts, and it uses data structures for off-line nearest common ancestors (in addition to the static tree DSU data structure of Gabow and Tarjan). In NRSS we made use of a hash table in order to store the CH values of the tree-edges and to be able to retrieve them efficiently. An alternative suggestion for this problem is to use radix sort [18]. However, the latter is only significant for the theoretical analysis, since in practice we observed that this part of the algorithm does not play a significant role in the total running time (as this it is dominated by the computation of other parameters). Furthermore, we made the following improvements in this algorithm. First, we handle the case of type-1 cuts using the same idea as in GIK and in our linear-time pointer-machine algorithm. Specifically, we detect those cuts by using the values $bcount$, $low1D$, $low1$, $low2D$ and $low2$. (This method is also suggested in the the deterministic algorithm of [18].) This confers an obvious advantage, since (1) these parameters can be computed easily and fast during the DFS, and (2) we thus avoid having to store $O(m)$ CH values and pointers to the corresponding back-edges. Second, we implemented a simpler and faster linear-time algorithm for computing all $MinDn$ and $MaxDn$ values, that uses only elementary data-structures. We observed that this algorithm is several times faster than the one suggested by [18], but it only slightly affects the total running-time, since that is dominated by the computation of $high$ and $highD$ points (i.e., the $MaxUp$ edges). Finally, we used two different algorithms for $MaxUp$, since the total running time is heavily dependent on the way this computation is performed. The first algorithm (as in GIK) simply traverses the adjacency lists. This is enough for GIK, but for NRSS we then have to sort the adjacency lists, in order to keep pointers to the $MaxUp$ edges, and this incurs a significant overhead. The second implementation of NRSS uses bucket-sort on the back-edges (as suggested by [18]), and it is called NRSS-sort. In both GIK and NRSS(-sort) we implemented the DSU data-structure by using path-compression and union-by-size. Although this is not theoretically optimal, in practice it works efficiently and we get a linear-time behaviour.

Experimental results for real-world graphs

For the experimental evaluation, we considered several real-world (undirected) graphs, taken from various application areas, as well as random graphs. For each graph G , we also compute a corresponding sparse 4-edge-connectivity certificate of G . A k -edge-connectivity certificate

of G is a spanning subgraph H of G such that, for any two vertices u and v and any positive integer $k' \leq k$, u and v are k' -edge-connected in H if and only if they are k' -edge-connected in G . Such a subgraph H has at most $k(n-1)$ edges and can be computed in linear time [20]. The reason why we use both the original graphs and their sparse certificates is twofold. First, we would like to observe if sparsity affects the relative performance of the algorithms. Second, we would like to see how more efficient is the computation of the 4-edge-connected components on the sparse certificates rather than on the original graphs.

Table 1 shows some statistics about the real-world graphs used in our experimental evaluation. The running times of the algorithms for the original graphs and for their sparse certificates are shown in Table 2 and Table 3 respectively.¹ Figure 2 and 3 illustrate the detailed running times for each part of the 4-edge-connected components algorithms on the real-world graphs and their sparse certificates. For each algorithm, it depicts the running time for computing the $(k-1)$ -edge-cuts and the corresponding k -connected components for $k \in \{1, 2, 3, 4\}$.

■ **Table 1** Characteristics of real-world graphs taken from the SNAP [16]; n is the number of vertices, m is the number of edges, and $\#k\text{CCs}$ is the number of k -edge-connected components; m' is the number of edges in a 4-edge-connectivity certificate of G . The network types are: social (SN), product (PN), collaboration, (CN), road (RN), and internet topology (IT).

Graph	n	m	$\#1\text{CCs}$	$\#2\text{CCs}$	$\#3\text{CCs}$	$\#4\text{CCs}$	m'	type
Deezer-HR	54573	498202	1	2436	5188	7878	194107	SN
Facebook-Artist	50515	819306	1	3220	6425	9144	179349	SN
com-Amazon	548552	925872	213690	244981	284574	335963	884127	PN
Gowalla	196591	950327	65294	112349	135493	145544	325636	SN
com-DBLP	425957	1049866	108878	155981	217842	268497	826192	CN
com-Youtube	1157828	2987624	22939	690029	860753	941713	1999435	SN
roadNet-CA	1971281	5533214	8713	8713	385230	385230	5520326	RN
twitch-gamers	168114	6797557	1	4081	7975	11800	648245	SN
as-Skitter	1696415	11095298	756	232897	493601	680010	5181377	IT
com-LiveJournal	3997962	34681189	38577	860464	1292384	1587631	2519167	SN
com-Orkut	3072627	117185083	187	67981	111261	149632	11953289	SN
com-Friendster	124836180	1806067135	59227815	73213653	79168479	82884698	203718281	SN

The algorithms GIK, Linear, NRSS and NRSS-sort compute the 4-edge-connected components, report the number of k -edge-connected components, for $k \leq 4$, and the corresponding times to compute them, as well as the number of minimal k -edge cuts, for $k \leq 3$. (We note that the number of minimal k -edge cuts can be as large as $O(n^2)$ and $O(n^3)$ for $k = 2, 3$, respectively.) For computing the k -edge-connected components for $k < 4$, all algorithms use the same routines. For $k = 1$ we simply perform a BFS; for $k = 2$ we implement the classic algorithm of Tarjan that uses the *low* points [23]. For $k = 3$ we use the algorithm of GK, which is the simplest and fastest known for computing 2-edge cuts and 3-edge-connected components [7, 10]. For every 3-edge-connected component that we compute, we construct the auxiliary 3-edge-connected graph that corresponds to it (which essentially maintains its k -edge cuts, for all $k \geq 3$) [2], by adding some virtual edges, and then we run the corresponding 3-edge cut algorithm on this graph. Finally, after we have computed all 3-edge cuts of each such graph (the number of those cuts is $O(n)$ in total), we compute its 4-edge-connected components by using the algorithm of [18] to compute the cactus tree.

¹ We note that we did not run the algorithms on the original com-Friendster graph, because its number of edges is too big for our implementations, due to our choice of variable types.

■ **Table 2** Running times in seconds of the algorithms for the real-world graphs of Table 1. For each algorithm we report the total running time and the time required to compute the 3-edge cuts and the corresponding 4-edge-connected components. Best running times for each graph are marked in bold.

Graph	NRSS-sort		NRSS		GIK		Linear	
	total	4ECCs	total	4ECCs	total	4ECCs	total	4ECCs
Deezer-HR	0.163	0.107	0.177	0.119	0.107	0.049	0.109	0.052
Facebook-Artist	0.223	0.153	0.229	0.158	0.128	0.059	0.136	0.066
com-Amazon	0.596	0.362	0.640	0.403	0.416	0.177	0.442	0.204
Gowalla	0.241	0.152	0.249	0.159	0.158	0.065	0.163	0.074
com-DBLP	0.533	0.322	0.575	0.362	0.374	0.158	0.387	0.174
com-Youtube	1.424	0.829	1.450	0.844	0.949	0.339	1.009	0.407
roadNet-CA	3.266	2.128	3.381	2.244	1.984	0.849	2.058	0.925
twitch-gamers	3.026	2.294	2.325	1.617	1.265	0.521	1.431	0.706
as-Skitter	5.676	3.918	5.717	3.961	3.321	1.561	3.645	1.883
com-LiveJournal	30.310	21.203	29.270	20.019	17.327	8.056	19.805	10.637
com-Orkut	94.435	70.280	85.745	61.550	43.171	19.172	55.081	30.316

From the running times reported in Tables 2 and 3, and plotted in Figures 2 and 3, we observe that running time of all algorithms is dominated by the computation of the 3-edge-cuts and the 4-edge-connected components. Indeed, computing the k -edge-connected components takes about twice as much as computing the $(k - 1)$ -edge-connected components. On average, the computation of the 3-edge-cuts and the 4-edge-connected components constitutes more than 60% of the running time of the NRSS algorithms and more than 40% of the running time of the GIK and Linear. Furthermore, as expected, all algorithms are executed faster on the sparse certificates, sometimes significantly, depending on the density of the original graph. (For instance, for twitch-gamers, all algorithms run about 7 times faster on the sparse certificate, while for com-LiveJournal, they run about 20 times faster on the sparse certificate.) Thus, for computing the 4-edge-connected components and the 3-cuts of a dense graph, it is more efficient to produce the sparse certificate (even with a straightforward algorithm that performs 4 passes over the edges of the graph) and apply any of those algorithms on it, than apply the algorithm directly on the original graph.

We observe that the running time of the Linear algorithm is very close to that of GIK. Indeed, GIK runs faster than Linear by about 7.5% on the original graphs and by about 3% on the sparse certificates. This means that the contraction operation does not incur a significant overhead in the total running time. One could expect the GIK algorithm to be worse than Linear, because it implements various sub-algorithms to handle the various subcases of type-3 cuts. However, all those sub-algorithms (except one, that needs an array of size $O(m)$) take time $O(n)$, because they rely on parameters already computed. Furthermore, although the Linear algorithm avoids the computation of *high* points, it uses as a replacement the $low_M D$ and low_M points, whose computation rely on a specific sorting of the adjacency lists.

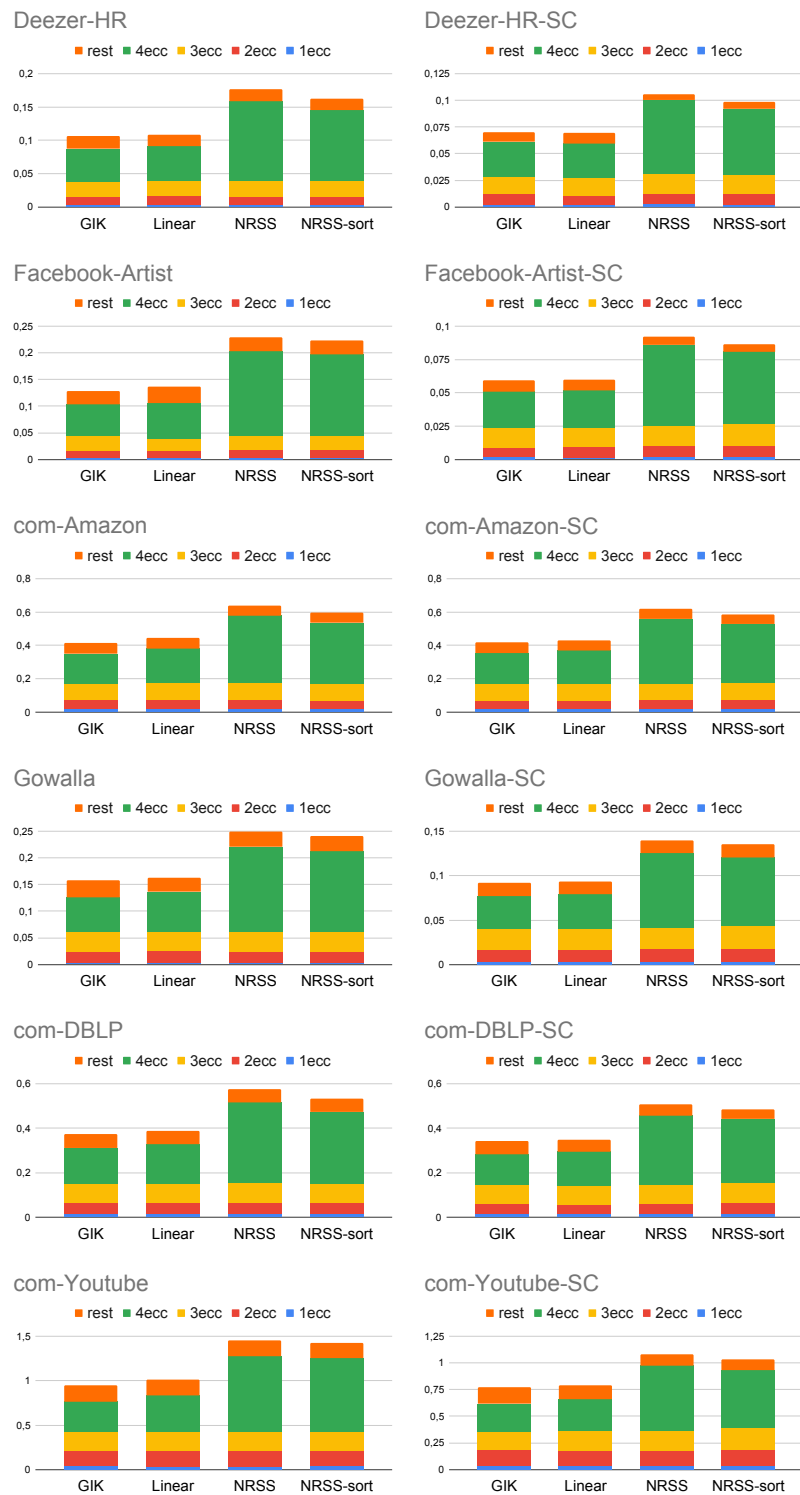
We also observe that the NRSS algorithms are consistently slower than GIK and Linear, by a significant factor, in both the original graphs and their sparse certificates. Specifically, GIK is about 40% faster than NRSS-sort on the original graphs, and about 35% faster on the sparse certificates. This is somewhat expected, because all these algorithms use the same parameters (or similar methods to compute those that they need), but the NRSS algorithms need also to maintain pointers to the edges that correspond to those parameters, and they also compute the CH values that use at least three RAM words for every edge.

■ **Table 3** Running times in seconds of the algorithms for the sparse certificates of the real-world graphs of Table 1. For each algorithm we report the total running time and the time required to compute the 3-edge cuts and the corresponding 4-edge-connected components. Best running times for each graph are marked in bold.

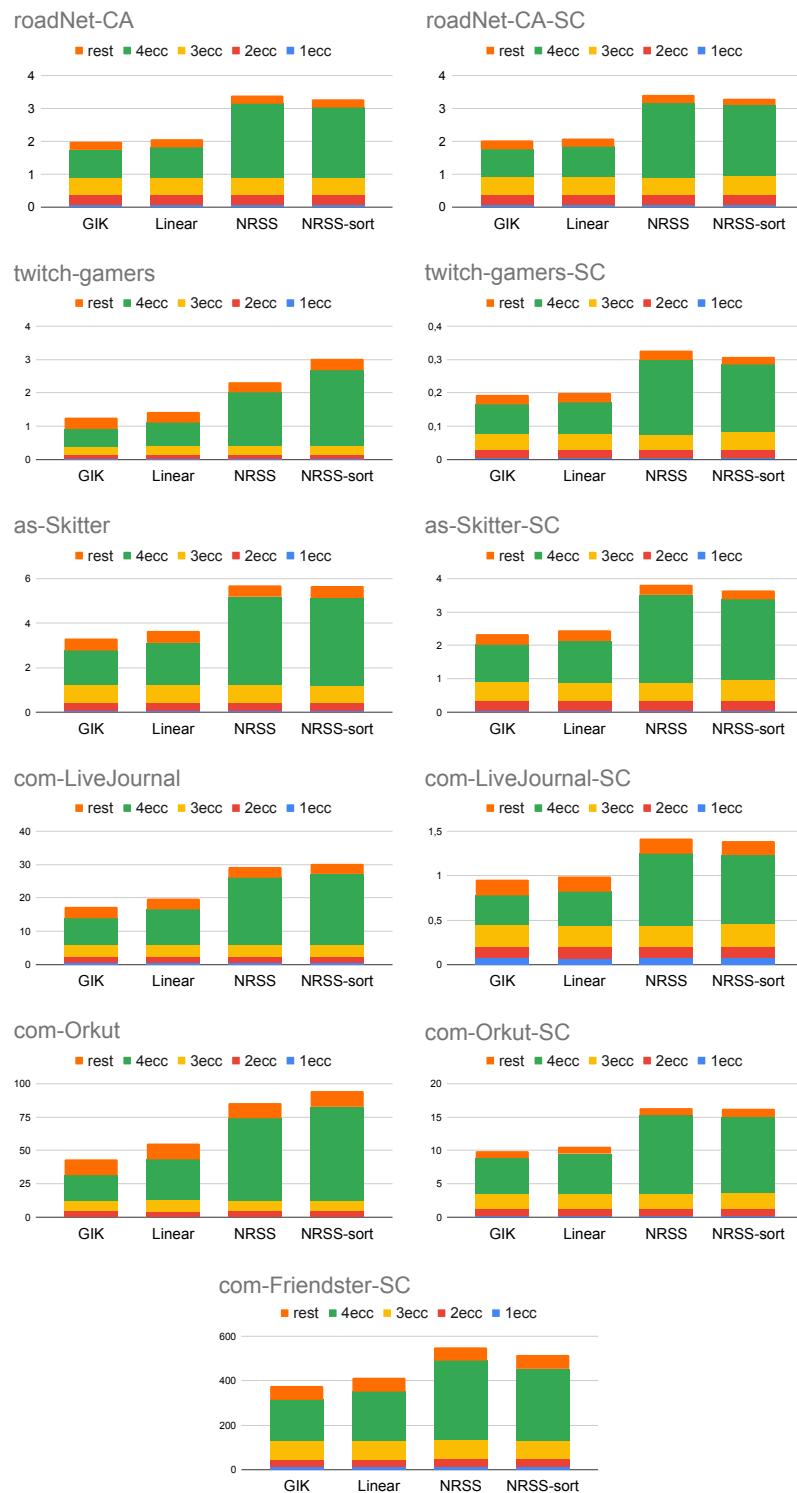
Graph	NRSS-sort		NRSS		GIK		Linear	
	total	4ECCs	total	4ECCs	total	4ECCs	total	4ECCs
Deezer-HR-SC	0.098	0.061	0.105	0.069	0.070	0.032	0.069	0.032
Facebook-Artist-SC	0.086	0.054	0.092	0.060	0.059	0.026	0.060	0.028
com-Amazon-SC	0.585	0.352	0.620	0.388	0.418	0.182	0.430	0.198
Gowalla-SC	0.124	0.070	0.133	0.079	0.093	0.037	0.093	0.038
com-DBLP-SC	0.486	0.288	0.508	0.312	0.343	0.141	0.349	0.153
com-Youtube-SC	1.037	0.544	1.082	0.591	0.770	0.260	0.791	0.295
roadNet-CA-SC	3.296	2.140	3.403	2.246	2.018	0.856	2.073	0.922
twitch-gamers-SC	0.307	0.203	0.326	0.221	0.193	0.088	0.200	0.096
as-Skitter-SC	3.643	2.432	3.830	2.614	2.336	1.106	2.445	1.236
com-LiveJournal-SC	1.389	0.773	1.419	0.811	0.958	0.338	0.992	0.383
com-Orkut-SC	16.216	11.422	16.386	11.789	9.943	5.342	10.597	6.014
com-Friendster-SC	515.693	323.119	551.180	357.877	376.956	184.727	413.305	220.934

Experimental results for random graphs

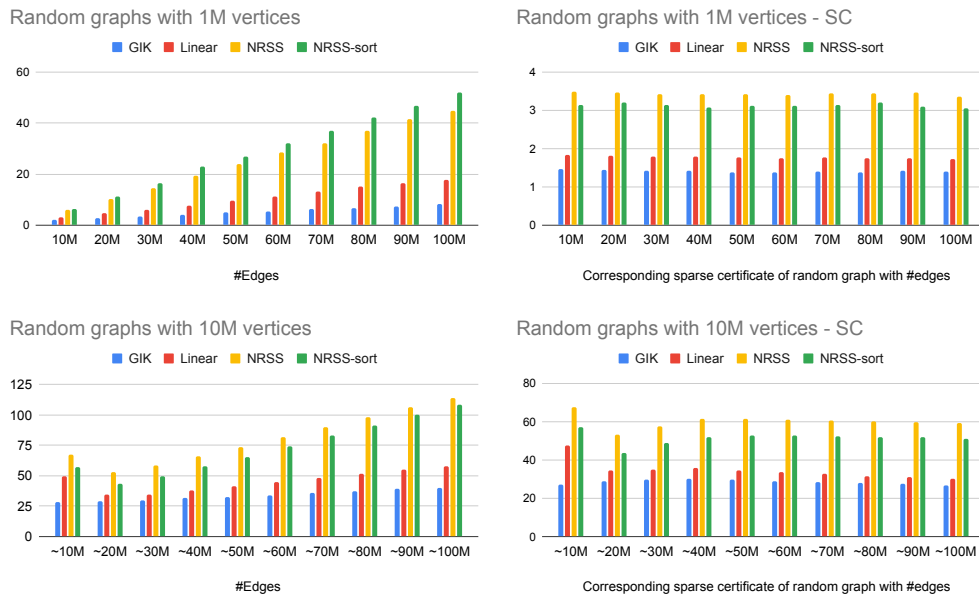
Finally, we explore further the relative performance of the four algorithms for computing 3-edge cuts (in 3-edge-connected graphs). Thus we created random graphs with a fixed number of vertices and an increasing number of edges, and we applied four algorithms on those graphs and on their sparse certificates. We augmented the graphs, whenever needed, in order to become 3-edge-connected, by adding a relatively small number of new edges. Specifically, we computed their connected components, and we connected them on a path. Then we applied the algorithm of Eswaran and Tarjan [4] to introduce the smallest number of edges that are needed to make them 2-edge-connected. And finally we applied the algorithm of Naor et al. [22] to optimally increase the connectivity by one. The corresponding plots are shown in Figure 4. In these experiments we notice that the GIK algorithm can be as much as 50% faster than Linear, and more than twice as fast as NRSS(-sort). This fact is not easily observable in the full algorithms for computing the 4-edge-connected components, because there are other unrelated operations taking place, and also the sizes of the 3-edge-connected components are most probably not large enough to make this difference manifest. We also notice that the NRSS-sort algorithm has, in fact, somewhat worse performance than NRSS on dense graphs.



■ **Figure 2** Detailed running times for each part of the 4-edge-connected components algorithms on the first six real-world graphs of Table 1 and their sparse certificates. For each algorithm, we plot the running time for computing the $(k-1)$ -edge-cuts and the corresponding k -connected components for $k \in \{1, 2, 3, 4\}$.



■ **Figure 3** Detailed running times for each part of the 4-edge-connected components algorithms on the last six real-world graphs of Table 1 and their sparse certificates. For each algorithm, we plot the running time for computing the $(k - 1)$ -edge-cuts and the corresponding k -connected components for $k \in \{1, 2, 3, 4\}$.



■ **Figure 4** Running times in seconds for random graphs with 1M and 10M vertices and their corresponding sparse certificates.

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)*, pages 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, pages 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 K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976. doi:10.1137/0205044.
- 5 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.
- 6 Z. Galil and G. F. Italiano. Reducing edge connectivity to vertex connectivity. *SIGACT News*, 22(1):57–61, March 1991. doi:10.1145/122413.122416.
- 7 L. Georgiadis, K. Giannis, G. F. Italiano, and E. Kosinas. Computing vertex-edge cut-pairs and 2-edge cuts in practice. In David Coudert and Emanuele Natale, editors, *19th International Symposium on Experimental Algorithms, SEA 2021, June 7-9, 2021, Nice, France*, volume 190 of *LIPICs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.SEA.2021.20.
- 8 L. Georgiadis, G. F. Italiano, and E. Kosinas. Computing the 4-edge-connected components of a graph in linear time. In *Proc. 29th European Symposium on Algorithms*, 2021. Full version available at <https://arxiv.org/abs/2105.02910>.
- 9 L. Georgiadis, G. F. Italiano, and E. Kosinas. Improved linear-time algorithm for computing the 4-edge-connected components of a graph, 2021. doi:10.48550/ARXIV.2108.08558.
- 10 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.

- 11 D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–55, 1984.
- 12 J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.
- 13 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.
- 14 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.
- 15 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*, pages 246–255, USA, 2009. Society for Industrial and Applied Mathematics.
- 16 J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- 17 K. Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.
- 18 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. Full version available at <https://arxiv.org/abs/2105.01699>.
- 19 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.
- 20 H. Nagamochi and T. Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, 2008. 1st edition.
- 21 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.
- 22 D. Naor, D. Gusfield, and C. Martel. A fast algorithm for optimally increasing the edge connectivity. *SIAM Journal on Computing*, 26(4):1139–1165, 1997.
- 23 R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
- 24 R. E. Tarjan. Finding dominators in directed graphs. *SIAM Journal on Computing*, 3(1):62–89, 1974.
- 25 R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.
- 26 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:<https://doi.org/10.1016/j.jda.2008.04.003>.




Algorithmic Meta-Theorems for Combinatorial Reconfiguration Revisited

Tatsuya Gima   

Nagoya University, Japan

Takehiro Ito   

Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Yasuaki Kobayashi   

Hokkaido University, Sapporo, Japan

Yota Otachi   

Nagoya University, Japan

Abstract

Given a graph and two vertex sets satisfying a certain feasibility condition, a reconfiguration problem asks whether we can reach one vertex set from the other by repeating prescribed modification steps while maintaining feasibility. In this setting, Mouawad et al. [IPEC 2014] presented an algorithmic meta-theorem for reconfiguration problems that says if the feasibility can be expressed in monadic second-order logic (MSO), then the problem is fixed-parameter tractable parameterized by treewidth + ℓ , where ℓ is the number of steps allowed to reach the target set. On the other hand, it is shown by Wrochna [J. Comput. Syst. Sci. 2018] that if ℓ is not part of the parameter, then the problem is PSPACE-complete even on graphs of bounded bandwidth.

In this paper, we present the first algorithmic meta-theorems for the case where ℓ is not part of the parameter, using some structural graph parameters incomparable with bandwidth. We show that if the feasibility is defined in MSO, then the reconfiguration problem under the so-called token jumping rule is fixed-parameter tractable parameterized by neighborhood diversity. We also show that the problem is fixed-parameter tractable parameterized by treedepth + k , where k is the size of sets being transformed. We finally complement the positive result for treedepth by showing that the problem is PSPACE-complete on forests of depth 3.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Combinatorial reconfiguration, monadic second-order logic, fixed-parameter tractability, treedepth, neighborhood diversity

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.61

Related Version *Full Version*: <https://arxiv.org/abs/2207.01024>

Funding *Takehiro Ito*: JSPS KAKENHI Grant Numbers JP18H04091, JP19K11814, JP20H05793

Yasuaki Kobayashi: JSPS KAKENHI Grant Numbers JP20H05793, JP20H00595, JP20K19742

Yota Otachi: JSPS KAKENHI Grant Numbers JP18H04091, JP20H05793, JP21K11752, JP22H00513

1 Introduction

A reconfiguration problem asks, given two feasible solutions S and S' of a combinatorial problem, whether there is a step-by-step transformation from S to S' without losing the feasibility [18]. The field studying such problems, called *combinatorial reconfiguration*, is growing rapidly. The source combinatorial problems in reconfiguration problems have spread in many subareas of theoretical computer science (see surveys [32, 38]). In this work, we focus on reconfiguration problems on graphs, especially the ones considering some vertex subsets as feasible solutions. Such problems involve classic properties like independent sets [19], vertex



© Tatsuya Gima, Takehiro Ito, Yasuaki Kobayashi, and Yota Otachi;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 61; pp. 61:1–61:15

Leibniz International Proceedings in Informatics



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

covers [28], dominating sets [35], and some connected variants [26]. Restrictions to some important graph classes such as bipartite graphs [24], split graphs [3], and sparse graphs [25] are also studied.

Since many problems are studied under many settings in combinatorial reconfiguration, one may ask for a unified method, or an *algorithmic meta-theorem*, for handling reconfiguration problems like Courcelle’s theorem for classic (non-reconfiguration) problems [1, 6, 9, 10, 12]. Since reconfiguration problems are hard in general (often PSPACE-complete [18]), we need to consider some special cases or introduce some additional parameters to consider fixed-parameter tractability. One successful approach in this direction was taken by Mouawad et al. [30], who showed that if the feasible solutions in a graph can be expressed in monadic second-order logic, then the reconfiguration problem (under reasonable transformation rules) is fixed-parameter tractable parameterized simultaneously by the treewidth of the underlying graph and the length of a transformation sequence. Their method is quite general and can be applied to several other settings.¹ On the other hand, Wrochna [40] showed that if the length of a transformation sequence is not part of the parameter, then some problems that fit in this framework are PSPACE-complete even on graphs of bounded bandwidth.

The two results mentioned above (the tractability parameterized by treewidth + transformation length [30] and the intractability parameterized solely by bandwidth [40]) might be interpreted as that if we have the length of a transformation sequence in the parameter, then we can do pretty much everything we expect, and otherwise we can expect very little. Thus, one might conclude that this line of research is complete and the length of a transformation sequence is necessary and sufficient in some sense for having efficient algorithms. Indeed, to the best of our knowledge, the study of meta-algorithms for reconfiguration problems was not extended after these results.

In this paper, we revisit the investigation of meta-algorithms for reconfiguration problems and shed light on the settings where the length of a transformation sequence is *not* part of the parameter. In particular, we present fixed-parameter algorithms for the reconfiguration problem of vertex sets defined by a monadic second-order formula parameterized by vertex cover number or neighborhood diversity. We also show that when combined with the solution set size, treedepth can be used to obtain a fixed-parameter algorithm. We then complement this result by showing that when the solution size is not part of the parameter, the problem is PSPACE-complete on graphs of constant treedepth.

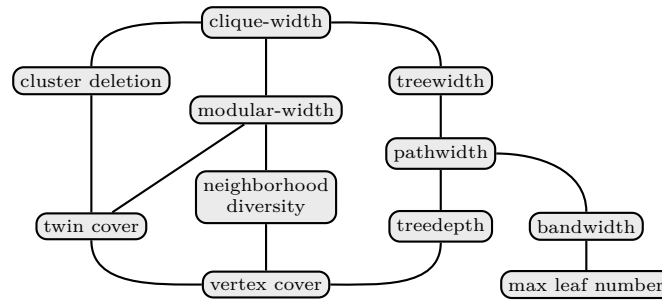
Due to the space limitation, the proofs of some results (marked with ★) are omitted or shortened. The full proofs will be included in the full version.

1.1 Our results

Now we give a little more precise description of our results. Formal definitions not given here can be found in Section 2 or in the full version.

For a graph G , we denote its clique-width by $\text{cw}(G)$, treewidth by $\text{tw}(G)$, treedepth by $\text{td}(G)$, vertex cover number by $\text{vc}(G)$, neighborhood diversity by $\text{nd}(G)$, cluster deletion number by $\text{cd}(G)$. (We define some of these parameters in the last part of Section 2.) See Figure 1 for the hierarchy among the graph parameters studied in this paper and some related ones. For a graph parameter f , we often say informally that a problem is fixed-parameter tractable “parameterized by f ” to mean “parameterized by $f(G)$, where G is the input graph.”

¹ We elaborate on this a little more in Section 1.2.



■ **Figure 1** The graph parameters studied in this paper. A connection between two parameters indicates the existence of a function in the one above that lower-bounds the one below.

Given a monadic second-order (MSO_1) formula ϕ with one free set variable, a graph G , and two vertex subsets S, S' of the same size, MSO_1 -RECONFIGURATION (MSO_1 -R) asks whether there exists a sequence of vertex subsets from S to S' such that each set in the sequence satisfies the property expressed by ϕ and each set in the sequence is obtained from the previous one by exchanging one vertex with another. Note that this rule allows to exchange any pair of vertices. Such a rule is well studied and called the *token jumping* rule [19]. There is another well-studied rule called the *token sliding* rule [17], which requires that the exchanged vertices are adjacent in G . We focus on the simpler rule token jumping in this paper and comment on the token sliding counter parts in the full version. MSO_2 -RECONFIGURATION (MSO_2 -R) with more general MSO_2 formulas is defined analogously.

To show a concrete example of MSO_1 -R, let $\phi(S) := \forall u \forall v: (u \in S \wedge v \in S) \Rightarrow \neg E(u, v)$. This ϕ is an MSO_1 formula (see Section 2) expressing that S is an independent set. Thus, MSO_1 -R with this ϕ is exactly INDEPENDENT SET RECONFIGURATION under the token jumping rule.

Now the main results in this paper can be summarized as follows.

1. MSO_1 -R is FPT parameterized by $\text{nd} + |\phi|$.
 - MSO_2 -R is FPT parameterized by $\text{vc} + |\phi|$, but not by $\text{nd} + |\phi|$ unless $\text{E} = \text{NE}$.
 - The positive results here strongly depend on the token jumping rule.
2. MSO_2 -R is FPT parameterized by $\text{td} + k + |\phi|$, where k is the size of input sets S and S' .
 - This result holds also under the token sliding rule.
 - As a by-product, we show that MSO_1 -R is FPT parameterized by $\text{cd} + k + |\phi|$. (Omitted in the short version.)
3. For some fixed ϕ , MSO_1 -R is PSPACE-complete even on forests of depth 3.
 - A similar hardness result can be shown under the token sliding rule.

In all positive results, we can find a shortest sequence for transformation (if any exists).

1.2 Related work

Wrochna [40] showed that MSO_1 -R is PSPACE-complete on graphs of constant bandwidth when ϕ expresses independent sets. This implies the PSPACE-completeness of MSO_1 -R on graphs of constant pathwidth, treewidth, and clique-width (see Figure 1). To cope with this intractability, Mouawad et al. [30] considered a variant with the additional restriction that the length of a transformation sequence cannot exceed some upper bound ℓ . They showed that this variant of MSO_2 -R is fixed-parameter tractable parameterized by $\ell + \text{tw} + |\phi|$. They reduce the reconfiguration problem to the model-checking problem of a single MSO_2 formula by expressing the existence of fewer than ℓ intermediate sets that satisfy ϕ and also expressing

that the change from a set to the next one obeys the transformation rule. Their framework is quite general and can be used in several other settings such as vertex sets defined by an MSO_1 formula with the parameter $\ell + \text{cw} + |\phi|$, or size- k vertex sets defined by a first-order formula with the parameter $\ell + k + |\phi|$ on a nowhere dense graph class (as observed also in [25]). Also, since the step-by-step modification can be defined by a formula, the results apply not only for the token jumping rule but also for several other rules including the token sliding rule.²

Another important line of studies on parameterized complexity of reconfiguration problems take the input set size k as the main parameter instead of a graph structural parameter. This line was initiated by Mouawad et al. [29], who showed several results parameterized solely by k and also by $k + \ell$. Recently, Bodlaender et al. [4, 5] further extended this line by showing that depending on whether and how the parameter depends on ℓ , the problem becomes complete to XL, XNL, or XNLP.

2 Preliminaries

We assume that the reader is familiar with the parameterized complexity theory. See a standard textbook (e.g., [13, 15, 16, 31]) for basic definitions.

Let $G = (V, E)$ be a graph. For $X \subseteq V$, we denote by $G[X]$ and $G - X$ the graphs induced by X and $V \setminus X$, respectively. We sometimes denote the vertex set of G by $V(G)$ and the edge set by $E(G)$. For a digraph D , we denote by $A(D)$ its arc set.

For a non-negative integer d , let $[d]$ denote the set $\{i \in \mathbb{Z} \mid 1 \leq i \leq d\}$. For two non-negative integers a, b with $a \leq b$, let $[a, b]$ denote the set $\{i \in \mathbb{Z} \mid a \leq i \leq b\}$.

Colored graphs. In this paper, we consider graphs in which each vertex has a (possibly empty) set of colors. We call them *colored graphs*. Formally, a colored graph G is a tuple (V, E, \mathcal{C}) such that the vertex set is V , the edge set $E \subseteq \binom{V}{2}$ is a set of unordered pairs of vertices, and $\mathcal{C} = \langle C_1, \dots, C_c \rangle$ is a tuple of subsets of V , where each C_i is called a *color*. For $v \in V$, let $\mathcal{C}(v)$ denote the set of colors that v belongs to. When $\mathcal{C}(v) = \emptyset$ for all $v \in V$, then the graph is *uncolored*. As we describe later, monadic second-order formulas treat the edge set as a symmetric binary relation on V and each color as a unary relation on V . As the number of colors a formula ϕ can access is bounded by $|\phi|$, which is always considered as a parameter or a constant in this paper, we can assume that the number of colors c is a parameter as well. We omit the information of colors and say $G = (V, E)$ when colors do not matter.

Two colored graphs $G = (V, E, \langle C_1, \dots, C_c \rangle)$ and $G' = (V', E', \langle C'_1, \dots, C'_c \rangle)$ are *isomorphic* if there is a color-preserving isomorphism $f: V \rightarrow V'$; that is,

- for all $u, v \in V$, $\{u, v\} \in E$ if and only if $\{f(u), f(v)\} \in E'$, and
- for all $v \in V$ and $1 \leq i \leq c$, $v \in C_i$ if and only if $f(v) \in C'_i$.

We also say that $\langle G, S \rangle$ and $\langle G', S' \rangle$ are isomorphic for sets $S \subseteq V$ and $S' \subseteq V'$ if the colored graphs $(V, E, \langle C_1, \dots, C_c, S \rangle)$ and $(V', E', \langle C'_1, \dots, C'_c, S' \rangle)$ are isomorphic.

Monadic second-order logic. In the monadic second-order logic on colored graphs, denoted MSO_1 , we can use vertex variables and vertex-set variables. The atomic formulas are the equality $x = y$ of vertex variables, the adjacency relation $E(x, y)$ which means $\{x, y\} \in E$, the color predicate $C_i(x)$ for each color C_i which means $x \in C_i$, and the inclusion predicate

² Actually, the rule used in [30] was another one called “token addition and removal.”

$X(x)$ for a variable x and a set variable X which means $x \in X$. The MSO_1 formulas are recursively defined from atomic formulas using the usual Boolean connectives (\neg , \wedge , \vee , \Rightarrow , \Leftrightarrow), and quantification of variables ($\forall x$, $\exists x$, $\forall X$, $\exists X$). For the sake of readability, we often use syntactic sugar in MSO_1 formulas (e.g., we write “ $\exists x \in X: \psi$ ” to mean “ $\exists x: X(x) \wedge \psi$ ”). As syntax sugars, we also use dotted quantifiers $\dot{\exists}$ and $\dot{\forall}$ to quantify distinct objects. For example, $\dot{\exists}a, b, c: \psi$ means $\exists a, b, c: (a \neq b) \wedge (b \neq c) \wedge (c \neq a) \wedge \psi$ and $\dot{\forall}a, b, c: \psi$ means $\forall a, b, c: ((a \neq b) \wedge (b \neq c) \wedge (c \neq a)) \Rightarrow \psi$.

MSO_2 is an extension of MSO_1 that additionally allows edge variables, edge-set variables, and an atomic formula $I(e, x)$ that represents the edge-vertex incidence relation. It is known that MSO_2 is strictly more powerful than MSO_1 in general [11].

An MSO_1 (or MSO_2) formula ϕ with free variables X_1, \dots, X_p is denoted by $\phi(X_1, \dots, X_p)$. For a graph G and vertex subsets S_1, \dots, S_p of G , we write $G \models \phi(S_1, \dots, S_p)$ if ϕ is true for G when the free variables X_1, \dots, X_p are interpreted as S_1, \dots, S_p . We call an MSO_1 (MSO_2) formula without free variables an MSO_1 (MSO_2 , resp.) *sentence*.

► **Proposition 2.1** (Folklore, see e.g., [23]). *Let G and G' be colored graphs and S and S' be some vertex subsets of them such that $\langle G, S \rangle$ and $\langle G', S' \rangle$ are isomorphic. Then, for every MSO_1 (or MSO_2) formula ϕ with one free set variable, $G \models \phi(S)$ if and only if $G' \models \phi(S')$.*

Problem definitions. For a colored graph G and an MSO_1 (or MSO_2) formula $\phi(X)$ a sequence S_0, \dots, S_ℓ of vertex subsets of G is a $\text{TJ}(\phi)$ -sequence from S_0 to S_ℓ (of length ℓ) if

- $|S_{i-1} \setminus S_i| = |S_i \setminus S_{i-1}| = 1$ for every $i \in [\ell]$, and
- $G \models \phi(S_i)$ for $0 \leq i \leq \ell$.

We denote by $\text{dist}_{\phi, G}(S, S')$ the minimum length of a $\text{TJ}(\phi)$ -sequence from S to S' , which is set to ∞ if there is no such sequence. We call a $\text{TJ}(\phi)$ -sequence of length 1 a $\text{TJ}(\phi)$ -move. Now the main problem studied in this paper can be formalized as follows.

MSO₁-RECONFIGURATION (MSO₁-R)

Input: An MSO_1 formula ϕ , a colored graph $G = (V, E, \mathcal{C})$, and sets $S, S' \subseteq V$ such that $|S| = |S'|$, $G \models \phi(S)$, and $G \models \phi(S')$.

Question: Is there a $\text{TJ}(\phi)$ -sequence from S to S' ?

We also study MSO_2 -R that allows MSO_2 formulas having one free vertex-set variable as ϕ . Observe that MSO_1 -R is PSPACE-hard as it generalizes various PSPACE-complete reconfiguration problems such as INDEPENDENT SET RECONFIGURATION. On the other hand, it still belongs to PSPACE since we can non-deterministically find the next vertex set R in the $\text{TJ}(\phi)$ -sequence and test whether $G \models \phi(R)$ holds in PSPACE [34, 39].

When describing a $\text{TJ}(\phi)$ -move from S_{i-1} to S_i , it is sometimes convenient to say that a *token* on the vertex $u \in S_{i-1} \setminus S_i$ is moved to the vertex $v \in S_i \setminus S_{i-1}$. The intuition behind this is that a vertex set in a $\text{TJ}(\phi)$ -sequence is considered as the positions of tokens and that in one $\text{TJ}(\phi)$ -move, token on some vertex jumps to another vertex. For simplicity, we sometimes write $S_{i-1} - u + v$ instead of $(S_{i-1} \setminus \{u\}) \cup \{v\}$.

Graph parameters. For a graph $G = (V, E)$, a set $S \subseteq V$ is a *vertex cover* if each $e \in E$ has at least one endpoint in S . The *vertex cover number* of G , denoted $\text{vc}(G)$, is the size of a minimum vertex cover of G . A vertex cover of size k of an n -vertex graph, if any exists, can be found in time $\mathcal{O}(c^k \cdot n)$ for some small constant c [8]. This implies that we can assume that a vertex cover of minimum size is given with the input when $\text{vc}(G)$ is part of the parameter.

Two vertices u and v are *twins* if $N(u) = N(v)$ or $N[u] = N[v]$. The *neighborhood diversity* of a graph G , denoted $\text{nd}(G)$, is the number of subsets V_i in the unique partition V_1, \dots, V_p of V into maximal sets of twin vertices. It is known that the neighborhood diversity and the corresponding partition can be computed in linear time [27,37]. From the definitions, we can see that $\text{nd}(G) \leq 2^{\text{vc}(G)} + \text{vc}(G)$ for every graph G [21].

The *treedepth* of a graph $G = (V, E)$, denoted $\text{td}(G)$, is the minimum depth d of a rooted forest F on the vertex set V such that each edge of G connects an ancestor and a descendant in F , where the depth of a forest is defined as the maximum distance between a root and a leaf. We call such a forest a *treedepth decomposition*. It is known a treedepth decomposition of depth d , if exists, can be found in time $2^{\mathcal{O}(d^2)} \cdot n$ [33]. Thus we may assume that a treedepth decomposition of depth $\text{td}(G)$ is given with the input when $\text{td}(G)$ is part of the parameter.

3 MSO₁-R parameterized by neighborhood diversity

The main result of this section is the following theorem.

► **Theorem 3.1.** *MSO₁-R parameterized by $\text{nd} + |\phi|$ is fixed-parameter tractable. Furthermore, for a yes instance of MSO₁-R, finding a shortest TJ(ϕ)-sequence is fixed-parameter tractable with the same parameter.*

We first prove Theorem 3.1 and then discuss the possibility of an extension to MSO₂-R. To prove Theorem 3.1, we first partition the feasible sets into a small number of equivalence classes. We show that the reachability between feasible sets can be checked by using an appropriately defined adjacency between the equivalence classes. Then, we take a deeper look at the connections between the equivalence classes and show that a shortest reconfiguration sequence can be found by finding some flow-like structure among the equivalence classes.

In the following, we fix the input of MSO₁-R as follows:

- $\phi(X)$: an MSO₁ formula with one free set variable X ;
 - $G = (V, E, \mathcal{C})$: a colored graph;
 - $S, S' \subseteq V$: the initial and target sets such that $G \models \phi(S)$, $G \models \phi(S')$, and $|S| = |S'| = k$.
- We say that a set $X \subseteq V$ is *feasible* if $G \models \phi(X)$.

We assume that the sets S and S' are colors in G ; that is, \mathcal{C} is of the form like $\langle C_1, \dots, C_c, S, S' \rangle$. If S and S' are originally not colors in G , we may add them and increase the number of colors only by 2.

Two vertices $u, v \in V$ in G are of the same *type* if u and v are twins and $\mathcal{C}(u) = \mathcal{C}(v)$. Let $\langle V_1, \dots, V_t \rangle$ be the partition of V into the sets of vertices of the same type. We call each V_i a *type*. Note that the type partition can be computed in polynomial time and that t depends only on the neighborhood diversity of G and the number of colors in \mathcal{C} .

For an MSO₁ formula ψ , let $\mathbf{q}(\psi) = 2^{\mathbf{q}_s} \cdot \mathbf{q}_v$, where \mathbf{q}_s and \mathbf{q}_v are the numbers of set and vertex quantifiers in ψ , respectively. Lampis [21] proved the following fact, which is one of the main ingredients in our algorithm.³

³ Note that Proposition 3.2 implies that, when neighborhood diversity is part of the parameter, the MSO₁ model-checking problem admits a small induced subgraph of the input graph as a kernel [21]. However, this does not directly show the fixed-parameter tractability of MSO₁-R (let alone the stronger claim of Theorem 3.1). In fact, as we will see later, an analogous result (Proposition 4.3) that implies a “natural” kernel for the MSO₁ model-checking problem parameterized by treedepth is known, while MSO₁-R is PSPACE-complete on graphs of constant treedepth (Theorem 5.1).

► **Proposition 3.2** ([21]). *Let ψ be an MSO_1 sentence. Assume that a graph H has more than $\mathfrak{q}(\psi)$ vertices of the same type, and H' is the graph obtained from H by removing a vertex in that type. Then, $H \models \psi$ if and only if $H' \models \psi$.*

We need the concept of “shapes” of vertex subsets that was used with Proposition 3.2 in the context of extended MSO_1 model-checking problems [20]. Here we introduce it in the following simplified form, which is sufficient for our purpose. The *signature* of $X \subseteq V$ is the mapping $\sigma_X: [t] \rightarrow \mathbb{Z}_{\geq 0}$ such that $\sigma_X(i) = |V_i \cap X|$. A *shape* is a mapping from $[t]$ to $\mathbb{Z}_{\geq 0} \cup \{\perp\}$ that maps each $i \in [t]$ to an element of $[0, \mathfrak{q}(\phi) - 1] \cup \{\perp\} \cup [|\mathbb{V}_i| - \mathfrak{q}(\phi) + 1, |\mathbb{V}_i|]$. Note that the number of shapes is $(2\mathfrak{q}(\phi) + 1)^t$. A set $X \subseteq V$ has shape $\bar{\sigma}$ if for every $i \in [t]$,

$$\bar{\sigma}(i) = \begin{cases} \perp & \mathfrak{q}(\phi) \leq \sigma_X(i) \leq |\mathbb{V}_i| - \mathfrak{q}(\phi), \\ \sigma_X(i) & \text{otherwise.} \end{cases}$$

We say that a shape $\bar{\sigma}$ is *k-feasible* if there is a feasible set $X \subseteq V$ of size k that has $\bar{\sigma}$ as its shape.

Let $\bar{\sigma}_S$ and $\bar{\sigma}_{S'}$ be the shapes of the input sets S and S' , respectively. Since S is a color of G , each type V_i either is a subset of S or has no intersection with S , that is,

$$\bar{\sigma}_S(i) = \begin{cases} |\mathbb{V}_i| & V_i \subseteq S, \\ 0 & \text{otherwise.} \end{cases}$$

This implies that a set $R \subseteq V$ has shape $\bar{\sigma}_S$ if and only if $R = S$. This applies to S' as well.

► **Observation 3.3.** *$R \subseteq V$ has shape $\bar{\sigma}_S$ ($\bar{\sigma}_{S'}$) if and only if $R = S$ ($R = S'$, resp.).*

Proposition 3.2 and the definition of shapes together give the following fact, which is known in more general forms in the previous studies (see e.g., [20]). This less general one is sufficient in our setting. We present a full proof in the full version to be self contained.

► **Lemma 3.4** (★). *If $R, R' \subseteq V$ have the same shape, then $G \models \phi(R)$ if and only if $G \models \phi(R')$.*

Lemma 3.4 implies in particular that if a shape $\bar{\sigma}$ is *k-feasible*, then every size- k set of shape $\bar{\sigma}$ is feasible.

► **Lemma 3.5** (★). *If feasible sets $R, R' \subseteq V$ have the same shape and size, then there is a $\text{TJ}(\phi)$ -sequence of length $|R \setminus R'|$ from R to R' such that all sets in the sequence have the same shape.*

We now introduce the *adjacency* between shapes. Intuitively, this concept captures how a single token jump connects different shapes. Let S_1 and S_2 be sets having different shapes $\bar{\sigma}_1$ and $\bar{\sigma}_2$, respectively, such that $S_1 \setminus S_2 = \{u\}$, $u \in V_i$, $S_2 \setminus S_1 = \{v\}$, $v \in V_j$, and $i \neq j$. For $h \in [t] \setminus \{i, j\}$, $\bar{\sigma}_1(h) = \bar{\sigma}_2(h)$ holds. Since $\bar{\sigma}_1 \neq \bar{\sigma}_2$, at least one of $\bar{\sigma}_1(i) \neq \bar{\sigma}_2(i)$ and $\bar{\sigma}_1(j) \neq \bar{\sigma}_2(j)$ holds. If $\bar{\sigma}_1(i) \neq \bar{\sigma}_2(i)$, then $|V_i \cap S_2| = |V_i \cap S_1| - 1$ implies that one of the following holds:

- A1: $\bar{\sigma}_1(i) \neq \perp$, $\bar{\sigma}_2(i) \neq \perp$, and $\bar{\sigma}_2(i) = \bar{\sigma}_1(i) - 1$;
- A2: $\bar{\sigma}_1(i) = \perp$ and $\bar{\sigma}_2(i) = \mathfrak{q}(\phi) - 1$; $(\sigma_{S_1}(i) = \mathfrak{q}(\phi))$
- A3: $\bar{\sigma}_1(i) = |\mathbb{V}_i| - \mathfrak{q}(\phi) + 1$ and $\bar{\sigma}_2(i) = \perp$. $(\sigma_{S_2}(i) = |\mathbb{V}_i| - \mathfrak{q}(\phi))$

Similarly, if $\bar{\sigma}_1(j) \neq \bar{\sigma}_2(j)$, then $|V_j \cap S_2| = |V_j \cap S_1| + 1$ implies that one of the following holds:

- B1: $\bar{\sigma}_1(j) \neq \perp$, $\bar{\sigma}_2(j) \neq \perp$, and $\bar{\sigma}_2(j) = \bar{\sigma}_1(j) + 1$;
- B2: $\bar{\sigma}_1(j) = \mathfrak{q}(\phi) - 1$ and $\bar{\sigma}_2(j) = \perp$; $(\sigma_{S_2}(j) = \mathfrak{q}(\phi))$
- B3: $\bar{\sigma}_1(j) = \perp$ and $\bar{\sigma}_2(j) = |\mathbb{V}_j| - \mathfrak{q}(\phi) + 1$. $(\sigma_{S_1}(j) = |\mathbb{V}_j| - \mathfrak{q}(\phi))$

Given the observation above, we say that two k -feasible shapes $\bar{\sigma}_1$ and $\bar{\sigma}_2$ are *adjacent* if and only if the following three conditions are satisfied.

- (1) One of the following holds:
 - $\bar{\sigma}_1$ and $\bar{\sigma}_2$ disagree at exactly two indices i and j such that i satisfies one of A1, A2, A3 and j satisfies one of B1, B2, B3;
 - $\bar{\sigma}_1$ and $\bar{\sigma}_2$ disagree at exactly one index i satisfying one of A1, A2, A3, or j satisfying one of B1, B2, B3.
- (2) There exists a size- k set S_1 of shape $\bar{\sigma}_1$ such that
 - if i is defined in (1) and $\bar{\sigma}_1(i) = \mathbb{I}$, then $\sigma_{S_1}(i) = \mathbf{q}(\phi)$;
 - if j is defined in (1) and $\bar{\sigma}_1(j) = \mathbb{I}$, then $\sigma_{S_1}(j) = |V_j| - \mathbf{q}(\phi)$.
- (3) There exists a size- k set S_2 of shape $\bar{\sigma}_2$ such that
 - if i is defined in (1) and $\bar{\sigma}_2(i) = \mathbb{I}$, then $\sigma_{S_2}(i) = |V_i| - \mathbf{q}(\phi)$;
 - if j is defined in (1) and $\bar{\sigma}_2(j) = \mathbb{I}$, then $\sigma_{S_2}(j) = \mathbf{q}(\phi)$.

The *size- k shape graph* \mathcal{S}_k has the set of k -feasible shapes as its vertex set and the adjacency between the vertices (shapes) is as defined above.

► **Lemma 3.6 (★).** *Let $\bar{\sigma}_1$ and $\bar{\sigma}_2$ be two different shapes that are k -feasible. Then, $\bar{\sigma}_1$ and $\bar{\sigma}_2$ are adjacent in \mathcal{S}_k if and only if there exist size- k feasible sets S_1 and S_2 of shapes $\bar{\sigma}_1$ and $\bar{\sigma}_2$, respectively, with $|S_1 \setminus S_2| = |S_2 \setminus S_1| = 1$.*

Since $|S| = |S'| = k$, the reachability between them can be reduced to the reachability between their shapes in \mathcal{S}_k .

► **Lemma 3.7 (★).** *Let $\bar{\sigma}$ and $\bar{\sigma}'$ be the shapes of S and S' , respectively. There is a $\text{TJ}(\phi)$ -sequence from S to S' if and only if $\bar{\sigma}$ and $\bar{\sigma}'$ belong to the same connected component of \mathcal{S}_k .*

Lemma 3.7 implies that $\text{MSO}_1\text{-R}$ can be solved by checking that the shapes of the initial and target sets belong to the same connected component of \mathcal{S}_k . We now show that \mathcal{S}_k can be constructed efficiently.

► **Lemma 3.8 (★).** *Constructing \mathcal{S}_k is fixed-parameter tractable parameterized by $t + \mathbf{q}(\phi)$.*

The lemma above already implies that $\text{MSO}_1\text{-R}$ is fixed-parameter tractable parameterized by $\text{nd} + |\phi|$. To find a shortest $\text{TJ}(\phi)$ -sequence, we take a closer look at \mathcal{S}_k .

A sequence $\bar{\sigma}_0, \dots, \bar{\sigma}_q$ of shapes with $\bar{\sigma}_i \neq \bar{\sigma}_{i+1}$ for $0 \leq i < q$ is the *shape sequence* of a $\text{TJ}(\phi)$ -sequence if the $\text{TJ}(\phi)$ -sequence can be split into $q + 1$ subsequences such that all sets in the i th subsequence have shape $\bar{\sigma}_i$ for $0 \leq i \leq q$.

► **Lemma 3.9 (★).** *If there is a $\text{TJ}(\phi)$ -sequence from S to S' , then there is a shortest one such that the corresponding shape sequence forms a simple path in \mathcal{S}_k .*

Lemma 3.9 implies that for finding a shortest $\text{TJ}(\phi)$ -sequence from S to S' , it suffices to first guess a path in \mathcal{S}_k and then find a shortest $\text{TJ}(\phi)$ -sequence having the path as its shape sequence. Note that $|V(\mathcal{S}_k)| \leq (2\mathbf{q}(\phi) + 1)^t$ and thus the number of candidates for such shape sequences is upper bounded by a function depending only on $\mathbf{q}(\phi)$ and t . (Recall that t is the number of types in G .) Therefore, the following lemma completes the proof of Theorem 3.1.

► **Lemma 3.10 (★).** *Given a sequence $\bar{\sigma}_0, \dots, \bar{\sigma}_q$ of shapes such that $\bar{\sigma}_0 = \bar{\sigma}_S$ and $\bar{\sigma}_q = \bar{\sigma}_{S'}$, finding a shortest $\text{TJ}(\phi)$ -sequence with the shape sequence $\bar{\sigma}_0, \dots, \bar{\sigma}_q$ is fixed-parameter tractable parameterized by $t + \mathbf{q}(\phi)$.*

Proof. We reduce the problem to MINIMUM-COST CIRCULATION defined as follows. Let $D = (X, A)$ be a directed graph. We define $\delta^{\text{in}}(v) = \{a \in A \mid a = (u, v) \in A\}$ and $\delta^{\text{out}}(v) = \{a \in A \mid a = (v, u) \in A\}$. A function $f: A \rightarrow \mathbb{R}$ is a *circulation* if $f(\delta^{\text{in}}(v)) = f(\delta^{\text{out}}(v))$ for each $v \in X$, where $f(A') = \sum_{a \in A'} f(a)$ for $A' \subseteq A$. A circulation f is an *integer circulation* if $f(a)$ is an integer for each $a \in A$. Given a *cost function* $w: A \rightarrow \mathbb{Q}$, the *cost* of a circulation f is defined as $\text{cost}(f) = \sum_{a \in A} w(a)f(a)$. Now, given a directed graph $D = (X, A)$, a *demand function* $d: A \rightarrow \mathbb{Q}$, a *capacity function* $c: A \rightarrow \mathbb{Q}$, and a cost function $w: A \rightarrow \mathbb{Q}$, MINIMUM-COST CIRCULATION asks to find a circulation f minimizing $\text{cost}(f)$ under the condition that $d(a) \leq f(a) \leq c(a)$ for each $a \in A$. It is known that MINIMUM-COST CIRCULATION can be solved in strongly polynomial time, and if the demand d and the capacity c take integer values only, then a minimum-cost integer circulation is found [36].

Now we construct an instance of MINIMUM-COST CIRCULATION from the graph $G = (V, E, \mathcal{C})$, its type partition (V_1, \dots, V_t) , and the shape sequence $\bar{\sigma}_0, \dots, \bar{\sigma}_q$. We first construct $D = (X, A)$. The digraph D contains two special vertices s and s' , and q sets L_0, L_1, \dots, L_{q-1} of vertices such that $L_j = \{v_1^j, \dots, v_t^j\}$ for $0 \leq j \leq q-1$. Each L_j is a bidirectional clique (i.e., there is an arc for each ordered pair of vertices in L_j). For $1 \leq j \leq q-1$, D contains the matching $\{(v_i^{j-1}, v_i^j) \mid 1 \leq i \leq t\}$ from L_{j-1} to L_j . There are arcs from s to all vertices in L_0 and from all vertices in L_{q-1} to s' . Additionally, D contains the arc (s', s) . Each arc a in each clique L_j has demand $d(a) = 0$, capacity $c(a) = \infty$, and cost $w(a) = 1$. All other arcs have cost 0. We set $d((s', s)) = c((s', s)) = k$. For $i \in [t]$, we set $d((s, v_i^0)) = c((s, v_i^0)) = |V_i \cap S|$ ($= \bar{\sigma}_0(i)$) and $d((v_i^{q-1}, s')) = c((v_i^{q-1}, s')) = |V_i \cap S'|$ ($= \bar{\sigma}_q(i)$). For $i \in [t]$ and $j \in [q-1]$, we set

$$d((v_i^{j-1}, v_i^j)) = \begin{cases} \mathfrak{q}(\phi) & \bar{\sigma}_j(i) = \mathbb{I}, \\ \bar{\sigma}_j(i) & \text{otherwise,} \end{cases} \quad c((v_i^{j-1}, v_i^j)) = \begin{cases} |V_i| - \mathfrak{q}(\phi) & \bar{\sigma}_j(i) = \mathbb{I}, \\ \bar{\sigma}_j(i) & \text{otherwise.} \end{cases}$$

In the full version, we show that there exists a $\text{TJ}(\phi)$ -sequence of length at most p with the shape sequence $\bar{\sigma}_0, \dots, \bar{\sigma}_q$ from S to S' if and only if the instance $\langle D, d, c, w \rangle$ of MINIMUM-COST CIRCULATION admits an integer circulation f of cost at most p . This completes the proof since MINIMUM-COST CIRCULATION is solvable in strongly polynomial time and the size of D depends only on t and the number of shapes. \blacktriangleleft

Using Lemma 6 in [21] and Theorem 3.1, we can show the following.

► **Corollary 3.11 (★).** *MSO₂-R parameterized by $\text{vc} + |\phi|$ is fixed-parameter tractable. Furthermore, for a yes instance of MSO₂-R, finding a shortest TJ-sequence is fixed-parameter tractable with the same parameter.*

On the other hand, using a hardness result in [22], we can show that an extension of Theorem 3.1 to MSO₂ is not possible under some reasonable assumption. Recall that $\text{E} = \text{DTIME}(2^{O(n)})$ and $\text{NE} = \text{NTIME}(2^{O(n)})$.

► **Theorem 3.12 (★).** *Unless $\text{E} = \text{NE}$, MSO₂-R on n -vertex uncolored graphs of neighborhood diversity 2 and twin cover number 3 cannot be solved in time $\mathcal{O}(n^{f(|\phi|)})$ for any function f .*

4 Fixed-parameter algorithm parameterized by the solution size and treedepth

In this section, we show that MSO₂-R is fixed-parameter tractable when parameterized simultaneously by treedepth, the length of the MSO₂ formula, and the size of input sets S and S' .

► **Theorem 4.1.** *MSO₂-R parameterized by $\text{td} + k + |\phi|$ is fixed-parameter tractable, where k is the size of input sets. Furthermore, for a yes instance of MSO₂-R, finding a shortest TJ-sequence is fixed-parameter tractable with the same parameter.*

As we show in Section 5, having the size of input sets is necessary since otherwise it is PSPACE-complete.

It is known (see e.g., [11]) that given a colored graph G and an MSO₂ sentence ϕ , one can compute in polynomial time a colored graph G' and an MSO₁ sentence ϕ' such that

- $G \models \phi$ if and only if $G' \models \phi'$;
- G' is obtained from G by subdividing each edge, and consider the new vertices introduced by the subdivisions as a color;
- the length of ϕ' is bounded by a function of $|\phi|$.

Observe that $\text{td}(G') \leq \text{td}(G) + 1$.⁴ Therefore, to prove Theorem 4.1, it suffices to show that MSO₁-R is fixed-parameter tractable parameterized by the claimed parameter.

Now we generalize the type of a vertex used in Section 3 to the type of a vertex set. For a colored graph $G = (V, E, \mathcal{C})$ and vertex sets $X, X' \subseteq V$, we say that X and X' have the same *type* if there is an isomorphism η from G to itself such that $\eta(X) = X'$, $\eta(X') = X$, and $\eta(v) = v$ for every $v \notin X \cup X'$. Note that from the definition of isomorphisms between colored graphs, $\mathcal{C}(v) = \mathcal{C}(\eta(v))$ holds for every $v \in V$. Note also that singletons $\{x\}, \{x'\} \subseteq V$ have the same type if and only if the vertices x and x' have the same type.

The next lemma says that if there are many disjoint vertex sets of the same type, then we can avoid most of them when finding TJ(ϕ)-sequences.

► **Lemma 4.2 (★).** *Let $\langle \phi, G, S, S' \rangle$ be a yes-instance of MSO₁-R with $|S| = |S'| = k$. Let C_1, \dots, C_t be a family of disjoint vertex sets with the same type not intersecting $S \cup S'$. If $t > k$, then for every $I \subseteq [t]$ with $|I| = k$, there is a shortest TJ(ϕ)-sequence S_0, \dots, S_ℓ from $S_0 = S$ to $S_\ell = S'$ such that $C_i \cap \bigcup_{0 \leq j \leq \ell} S_j \neq \emptyset$ only if $i \in I$.*

Next we further argue that if there are a much larger number of disjoint vertex sets of the same type, then we can safely remove some of them. Note that this claim is stronger than Lemma 4.2 in some sense. Since the formula $\phi(X)$ may depend on the whole structure of G (i.e., not only on $G[X]$), “not using it in a sequence” and “removing it from the graph” are different.

We need the following proposition, which is a generalization of Proposition 3.2.

► **Proposition 4.3** ([23]). *Let G be a colored graph and ϕ be an MSO₁ formula with one free set variable. Assume that G contains $t > 2^{p \cdot q(\phi)}$ disjoint size- p vertex sets with the same type. Let G' be the graph obtained from G by removing one of the t sets. Then, for every subset $X \subseteq V$ disjoint from the t sets, $G' \models \phi(X)$ if and only if $G \models \phi(X)$.*

Using Proposition 4.3, we can show the following.

► **Lemma 4.4 (★).** *Let $\langle \phi, G, S, S' \rangle$ be an instance of MSO₁-R with $|S| = |S'| = k$. Let C_1, \dots, C_t be a family of disjoint size- p vertex sets with the same type not intersecting $S \cup S'$. If $t > k + 2^{p \cdot q(\phi)}$, then for every $C \in \{C_1, \dots, C_t\}$, $\text{dist}_{\phi, G}(S, S') = \text{dist}_{\phi, G-C}(S, S')$.*

⁴ Starting with a treedepth decomposition F of G with depth at most d , we construct a treedepth decomposition F' of G' with depth at most $d + 1$ by adding the vertex corresponding to each edge $\{u, v\} \in E(G)$ as a leaf attached to one of u and v that is a descendant of the other.

The next lemma completes the proof of Theorem 4.1 as it means that we have a kernel of $\text{MSO}_1\text{-R}$ parameterized by $\text{td}(G) + k + |\phi|$ that preserves the minimum length of a $\text{TJ}(\phi)$ -sequence.

► **Lemma 4.5.** *Let $\langle \phi, G, S, S' \rangle$ be an instance of $\text{MSO}_1\text{-R}$ with $|S| = |S'| = k$. In polynomial time, one can compute a subgraph H of G such that $\text{dist}_{\phi, G}(S, S') = \text{dist}_{\phi, H}(S, S')$ and the size of H depends only on the parameter $\text{td}(G) + k + |\phi|$.*

Proof. Let F be a treedepth decomposition of depth $\text{td}(G)$. If F is not connected, then we add a new vertex r and add edges from the new vertex to the roots of trees in F and set r to the new root. We call the resultant tree T . If F is connected, then we just set $T = F$ and call its root r . Let d be the depth of T . Note that $d \leq \text{td}(G) + 1$.

A node in T has *height* h if the maximum distance to a descendant is h , where the height of a leaf is 0. Let $c(0) = 0$, $n(0) = 1$, and for $h \geq 0$, let

$$\begin{aligned} c(h+1) &= (k + 2^{n(h) \cdot q(\phi)}) \cdot 2^{|\phi| \cdot n(h)} \cdot 2^{(n(h)+d-h)^2} + 2k, \\ n(h+1) &= n(h) \cdot c(h+1) + 1. \end{aligned}$$

In the next paragraph, we show that after exhaustively applying Lemma 4.4 in a bottom-up manner along T , each node of height h has at most $c(h)$ children and each subtree rooted at a node of height h contains at most $n(h)$ nodes. This implies that H has at most $n(d)$ vertices, where $n(d)$ depends only on $\text{td}(G)$, k , and $|\phi|$. If $h = 0$, then the claim is trivial. Assume that the claim holds for some $h \geq 0$. It suffices to prove the upper bound $c(h+1)$ for the number of children as the upper bound $n(h+1)$ follows immediately. Suppose to the contrary that a node v of height $h+1$ has more than $c(h+1)$ children. Since $|S \cup S'| \leq 2k$, more than $c(h+1) - 2k$ subtrees rooted at the children of v have no intersection with $S \cup S'$. Let S_1, \dots, S_p be such subtrees. By the induction hypothesis, $|V(S_i)| \leq n(h)$ holds for $i \in [p]$. Let R be the vertices on the v - r path in T (including v and r). Observe that, in H , the vertices in $V(S_i)$ may have neighbors only in $V(S_i) \cup R$. Thus the number of different types of $V(S_1), \dots, V(S_p)$ is at most $2^{|\phi| \cdot n(h)} \cdot 2^{(n(h)+d-h)^2}$, where $2^{|\phi| \cdot n(h)}$ is the number of possible ways for coloring $n(h)$ vertices with subsets of at most $|\phi|$ colors and $2^{(n(h)+d-h)^2}$ is an upper bound on the number of different ways that $n(h)$ vertices form a graph and have additional neighbors in $d-h$ vertices. Since $p > c(h+1) - 2k = (k + 2^{n(h) \cdot q(\phi)}) \cdot 2^{|\phi| \cdot n(h)} \cdot 2^{(n(h)+d-h)^2}$, there is a subset $I \subseteq [p]$ such that $|I| > k + 2^{n(h) \cdot q(\phi)}$ and all vertex sets $V(S_i)$ with $i \in [I]$ have the same type. This is a contradiction as Lemma 4.4 can be applied here.

Finally, let us see how fast we can apply Lemma 4.4 exhaustively in a bottom-up manner. The description above immediately gives a fixed-parameter algorithm parameterized by $\text{td}(G) + k + |\phi|$, which is actually sufficient for our purpose. A polynomial-time algorithm can be achieved in pretty much the same way as presented in [14] for a reconfiguration problem of paths parameterized by td . The idea is to use a polynomial-time algorithm for labeled-tree isomorphism to classify subtrees into different types. Only the differences here are that the graph is colored and the parameters involved are larger. As the colors of the vertices can be handled by a labeling algorithm and the involved parameters do not matter when they are too large (i.e., if it is $|V|$ or more), we still obtain a polynomial-time algorithm. ◀

5 PSPACE-completeness on forests of depth 3

In this section, we complement Theorem 4.1 by showing that if the size of input sets is not part of the parameter, then the problem becomes PSPACE-complete.

For a set U , a subset family $\mathcal{C} \subseteq 2^U$ is an *exact cover* if the elements of \mathcal{C} are pairwise disjoint and $\bigcup_{C \in \mathcal{C}} C = U$. For two exact covers $\mathcal{C}_1, \mathcal{C}_2$ of U , we say that \mathcal{C}_1 can be obtained from \mathcal{C}_2 by a *merge* (and \mathcal{C}_2 can be obtained from \mathcal{C}_1 by a *split*) if $\mathcal{C}_1 \setminus \mathcal{C}_2 = \{D_1\}$ and $\mathcal{C}_2 \setminus \mathcal{C}_1 = \{D_2, D_3\}$ for some D_1, D_2, D_3 . Note that $D_1 = D_2 \cup D_3$ and $D_2 \cap D_3 = \emptyset$ as \mathcal{C}_1 and \mathcal{C}_2 are exact covers.

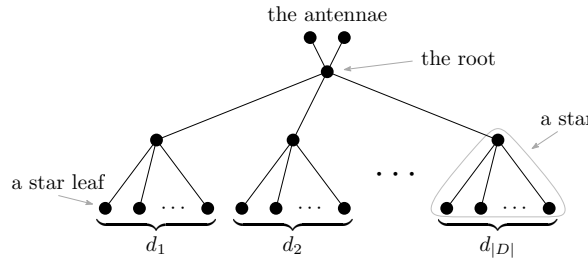
Given a set U , a family $\mathcal{D} \subseteq 2^U$, and two exact covers $\mathcal{C}, \mathcal{C}' \subseteq \mathcal{D}$ of U , EXACT COVER RECONFIGURATION asks whether there exists a sequence $\mathcal{C}_0, \dots, \mathcal{C}_\ell$ of exact covers of U from $\mathcal{C} = \mathcal{C}_0$ to $\mathcal{C}' = \mathcal{C}_\ell$ such that $\mathcal{C}_i \subseteq \mathcal{D}$ for all i and \mathcal{C}_i is obtained from \mathcal{C}_{i-1} by a split or a merge for each $i \in [\ell]$. It is known that EXACT COVER RECONFIGURATION is PSPACE-complete [7].

In this section, we prove the following hardness result by reducing EXACT COVER RECONFIGURATION to MSO₁-R. (Recall that MSO₁-R belongs to PSPACE.)

► **Theorem 5.1 (★).** *For some fixed ϕ , MSO₁-R is PSPACE-complete on uncolored forests of depth 3.*

Let $\langle U, \mathcal{D}, \mathcal{C}, \mathcal{C}' \rangle$ be an instance of EXACT COVER RECONFIGURATION. We construct an equivalent instance of MSO₁-R. Without loss of generality, we assume that U is a set of positive integers greater than or equal to 3.

For each set $D \in \mathcal{D}$, we construct a tree T_D as follows (see Figure 2). The tree T_D contains a central vertex called the *root*. For each $d \in D$, the root has a child with d grandchildren. We call the subtree rooted at a child of the root a *star* and each leaf in a star a *star leaf*. Additionally, the root has two more children that have degree 1. They are called the *antennae*.



■ **Figure 2** The tree T_D with $D = \{d_1, d_2, \dots, d_{|D|}\}$.

The entire forest F consists of trees T_D for all $D \in \mathcal{D}$ and eight isolated vertices. By I , we denote the set of the isolated vertices. Clearly, F has treedepth 3. The initial set S consists of all vertices in I and all star leaves of T_D for all $D \in \mathcal{C}$. Similarly, the target set S' consists of all vertices in I and all star leaves of T_D for all $D \in \mathcal{C}'$. Note that $|S| = |S'| = |U| + 8$.

For a set $R \subseteq V(F)$ and a set $D \in \mathcal{D}$, the tree T_D is *full (empty) under R* if R contains all (no, resp.) star leaves of T_D , and T_D is *clean* if it is full or empty. We also say that, for a set $R \subseteq V(F)$ and a set $D \in \mathcal{D}$, a star in T_D is *full (empty)* if R contains all (no, resp.) star leaves of the star, and the star is *clean* if it is full or empty.

A tree T_D is *marked* by a vertex set if both antennae are included in the vertex set. A star in T_D is *marked* by a vertex set if the center (i.e., the unique non-leaf vertex) of the star is included. We use the eight additional vertices to mark trees and stars.

We construct the MSO₁ formula $\phi(X)$ expressing that X satisfies 1 or 2 below.

1. All trees T_D are clean and exactly eight vertices in X are not star leaves.
2. Exactly three trees $T_{D_1}, T_{D_2}, T_{D_3}$ are marked, all other trees are clean, and the following conditions are satisfied.

- Exactly one of the three trees, say T_{D_3} , is clean.
- In each of T_{D_1} and T_{D_2} , exactly one star is marked and all other stars are clean.
- The marked star in T_{D_1} is clean if and only if so is the marked star in T_{D_2} .

Constructing such $\phi(X)$ is tedious but not difficult. The expression is given in the full version. Now it suffices to show that the constructed instance $\langle \phi, F, S, S' \rangle$ is a yes-instance of $\text{MSO}_1\text{-R}$ if and only if $\langle U, \mathcal{D}, \mathcal{C}, \mathcal{C}' \rangle$ is a yes-instance of $\text{EXACT COVER RECONFIGURATION}$. The rest of the proof is omitted in the short version.

6 Conclusion

We revisited the reconfiguration problems of vertex sets defined by MSO formulas, while putting the length constraint of reconfiguration sequence aside. We showed that the problem is fixed-parameter tractable parameterized solely by neighborhood diversity and by the combination of treedepth and the vertex-set size. The parameterization solely by treedepth would not work as we showed that the problem is PSPACE -complete on forests of depth 3.

Given the positive result for neighborhood diversity and the known hardness for clique-width (implied by the one for bandwidth [40]), a natural target would be an extension to modular-width, which is a parameter sitting between neighborhood diversity and clique-width (see Figure 1). It is known that a special case, the independent set reconfiguration, is fixed-parameter tractable parameterized by modular-width [2], but the algorithm in [2] is already quite nontrivial.

Another direction would be strengthening the hardness for treedepth. In Section 5, we showed the hardness for a quite complicated and rather unnatural formula ϕ , which simulates the merge and split operations. Although this rules out the possibility of meta-theorems parameterized by treedepth, it would be still interesting to investigate the complexity of specific more natural problems. For example, what is the complexity of the independent set reconfiguration and the dominating set reconfiguration parameterized solely by treedepth?

References

- 1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.
- 2 Rémy Belmonte, Tesshu Hanaka, Michael Lampis, Hirotaka Ono, and Yota Otachi. Independent set reconfiguration parameterized by modular-width. *Algorithmica*, 82(9):2586–2605, 2020. doi:10.1007/s00453-020-00700-y.
- 3 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory Comput. Syst.*, 65(4):662–686, 2021. doi:10.1007/s00224-020-09967-8.
- 4 Hans L. Bodlaender, Carla Groenland, Jesper Nederlof, and Céline M. F. Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *FOCS 2021*, pages 193–204. IEEE, 2021. doi:10.1109/FOCS52979.2021.00027.
- 5 Hans L. Bodlaender, Carla Groenland, and Céline M. F. Swennenhuis. Parameterized complexities of dominating and independent set reconfiguration. In *IPEC 2021*, volume 214 of *LIPICs*, pages 9:1–9:16, 2021. doi:10.4230/LIPICs.IPEC.2021.9.
- 6 Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(5&6):555–581, 1992. doi:10.1007/BF01758777.
- 7 Jean Cardinal, Erik D. Demaine, David Eppstein, Robert A. Hearn, and Andrew Winslow. Reconfiguration of satisfying assignments and subset sums: Easy to find, hard to connect. *Theor. Comput. Sci.*, 806:332–343, 2020. doi:10.1016/j.tcs.2019.05.028.

- 8 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.
- 9 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.
- 10 Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *RAIRO Theor. Informatics Appl.*, 26:257–286, 1992. doi:10.1051/ita/1992260302571.
- 11 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*. Cambridge University Press, 2012. URL: <https://www.cambridge.org/knowledge/isbn/item5758776/>.
- 12 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.
- 13 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 14 Erik D. Demaine, David Eppstein, Adam Hesterberg, Kshitij Jain, Anna Lubiw, Ryuhei Uehara, and Yushi Uno. Reconfiguring undirected paths. In *WADS 2019*, volume 11646 of *Lecture Notes in Computer Science*, pages 353–365, 2019. doi:10.1007/978-3-030-24766-9_26.
- 15 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 16 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.
- 17 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.
- 18 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.
- 19 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.
- 20 Dusan Knop, Martin Koutecký, Tomáš Masarík, and Tomáš Toufar. Simplified algorithmic metatheorems beyond MSO: treewidth and neighborhood diversity. *Log. Methods Comput. Sci.*, 15(4), 2019. doi:10.23638/LMCS-15(4:12)2019.
- 21 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.
- 22 Michael Lampis. Model checking lower bounds for simple graphs. *Log. Methods Comput. Sci.*, 10(1), 2014. doi:10.2168/LMCS-10(1:18)2014.
- 23 Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. In *ISAAC 2021*, volume 212 of *LIPICs*, pages 34:1–34:15, 2021. doi:10.4230/LIPICs.ISAAC.2021.34.
- 24 Daniel Lokshtanov and Amer E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms*, 15(1):7:1–7:19, 2019. doi:10.1145/3280825.
- 25 Daniel Lokshtanov, Amer E. Mouawad, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. *J. Comput. Syst. Sci.*, 95:122–131, 2018. doi:10.1016/j.jcss.2018.02.004.
- 26 Daniel Lokshtanov, Amer E. Mouawad, Fahad Panolan, and Sebastian Siebertz. On the parameterized complexity of reconfiguration of connected dominating sets. *Algorithmica*, 84(2):482–509, 2022. doi:10.1007/s00453-021-00909-5.
- 27 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discret. Math.*, 201(1-3):189–241, 1999. doi:10.1016/S0012-365X(98)00319-7.

- 28 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Sebastian Siebertz. Vertex cover reconfiguration and beyond. *Algorithms*, 11(2):20, 2018. doi:10.3390/a11020020.
- 29 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017. doi:10.1007/s00453-016-0159-2.
- 30 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Marcin Wrochna. Reconfiguration over tree decompositions. In *IPEC 2014*, volume 8894 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2014. doi:10.1007/978-3-319-13524-3_21.
- 31 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. doi:10.1093/ACPROF:OSO/9780198566076.001.0001.
- 32 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018. doi:10.3390/a11040052.
- 33 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In *ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942, 2014. doi:10.1007/978-3-662-43948-7_77.
- 34 Larry J. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Department of Electrical Engineering, MIT, 1974. URL: <http://hdl.handle.net/1721.1/15540>.
- 35 Akira Suzuki, Amer E. Mouawad, and Naomi Nishimura. Reconfiguration of dominating sets. *J. Comb. Optim.*, 32(4):1182–1195, 2016. doi:10.1007/s10878-015-9947-x.
- 36 Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Comb.*, 5(3):247–256, 1985. doi:10.1007/BF02579369.
- 37 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *ICALP 2008*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. doi:10.1007/978-3-540-70575-8_52.
- 38 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CB09781139506748.005.
- 39 Moshe Y. Vardi. The complexity of relational query languages. In *STOC 1982*, pages 137–146, 1982. doi:10.1145/800070.802186.
- 40 Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *J. Comput. Syst. Sci.*, 93:1–10, 2018. doi:10.1016/j.jcss.2017.11.003.

Efficient Recognition of Subgraphs of Planar Cubic Bridgeless Graphs

Miriam Goetze  

Karlsruhe Institute of Technology, Germany

Paul Jungeblut  

Karlsruhe Institute of Technology, Germany

Torsten Ueckerdt 

Karlsruhe Institute of Technology, Germany

Abstract

It follows from the work of Tait and the Four-Color-Theorem that a planar cubic graph is 3-edge-colorable if and only if it contains no bridge. We consider the question of which planar graphs are subgraphs of planar cubic bridgeless graphs, and hence 3-edge-colorable. We provide an efficient recognition algorithm that given an n -vertex planar graph, augments this graph in $\mathcal{O}(n^2)$ steps to a planar cubic bridgeless supergraph, or decides that no such augmentation is possible. The main tools involve the GENERALIZED (ANTI)FACTOR-problem for the fixed embedding case, and SPQR-trees for the variable embedding case.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases edge colorings, planar graphs, cubic graphs, generalized factors, SPQR-tree

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.62

Related Version *Full Version:* <https://arxiv.org/abs/2204.11750> [13]

1 Introduction

Whether or not the 3-EDGE COLORABILITY-problem is solvable in polynomial time for planar graphs is one of the most fundamental open problems in algorithmic graph theory:

► **Question 1.** *Can we decide in polynomial time, whether the edges of a given planar graph can be colored in three colors such that any two adjacent edges receive distinct colors?*

In other words, can we decide for a planar graph G in polynomial time whether $\chi'(G) \leq 3$, where $\chi'(G)$ denotes the chromatic index of G ? Clearly, it is enough to consider planar graphs G of maximum degree $\Delta(G) = 3$. If G is planar and 3-regular, then by the Four-Color-Theorem [1, 2] and the work of Tait [26] we know that G is 3-edge-colorable if and only if G is bridgeless. An edge is a bridge if its removal increases the number of connected components (note that this definition also applies to disconnected graphs). As we can check the existence of bridges in linear time [28], we hence can decide in polynomial time whether a given 3-regular planar graph is 3-edge-colorable.

In particular, subgraphs of bridgeless 3-regular planar graphs are 3-edge-colorable. However, this does not answer Question 1 yet (as sometimes wrongly claimed, e.g., in [7]), because it is for example not clear which planar graphs of maximum degree 3 are subgraphs of bridgeless 3-regular planar graphs, and whether these can be recognized efficiently.

In this paper we consider the corresponding decision problem: Given a graph G , is there a bridgeless 3-regular planar graph H , such that $G \subseteq H$? In other words, can G be augmented, by adding edges and (possibly) vertices, to a supergraph H of G that is planar, 3-regular, and contains no bridge? For brevity we call such a supergraph H a *3-augmentation* of G and denote the above decision problem as 3-AUGMENTATION. Our main result is that 3-AUGMENTATION is in P.



© Miriam Goetze, Paul Jungeblut, and Torsten Ueckerdt;
licensed under Creative Commons License CC-BY 4.0

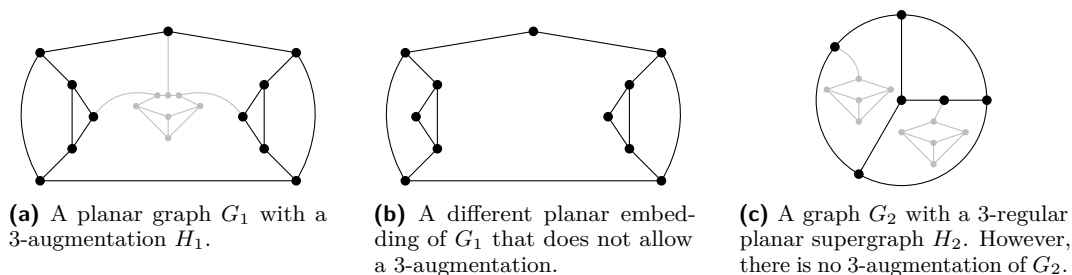
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 62; pp. 62:1–62:14



Leibniz International Proceedings in Informatics

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



■ **Figure 1** Example instances for the 3-AUGMENTATION problem.

► **Theorem 2.** *For a given n -vertex graph G we can construct in $\mathcal{O}(n^2)$ time a 3-regular bridgeless planar supergraph H of G , or conclude that no such exists.*

Theorem 2 is the main result of the present paper and we emphasize that this does not answer Question 1 yet. In fact, admitting a 3-augmentation is a sufficient condition for 3-edge colorability; but it is in general not necessary. For example, $K_{2,3}$ admits a proper 3-edge coloring but no 3-augmentation. Question 1 remains open and we discuss it and its connection to 3-augmentations in more detail in Section 3.

In order to decide the existence of a 3-augmentation (i.e., proving Theorem 2), we may of course assume that the graph G itself is planar and of maximum degree at most 3. Observe that in this case it is always possible to find a 3-regular planar supergraph of G , for example by adding the small gadget $K_4^{(1)}$ consisting of K_4 with one subdivided edge to each vertex that has not degree 3 yet, see Figure 1c. The difficult part is to prevent bridges in the resulting graph, even if the input graph G is already bridgeless. In fact, our task boils down to finding a suitable planar embedding of G such that for each vertex v of G and each missing edge at v , we can assign an incident face at v that should contain the new edge. We avoid the creation of bridges by assigning each face either no or at least two such new edges. Having assigned k new edges to a face f , we insert the small gadget $K_4^{(k)}$ consisting of K_4 with one edge subdivided k times into f . See Figure 1a for an example. Let us note that this might only work for some planar embeddings of G . See Figure 1b for a negative example.

We show Theorem 2 in three steps. First, we show that G admits a 3-augmentation if and only if each inclusion-maximal 2-connected component, called a *block*, of G admits a 3-augmentation. As all blocks can be found in linear time [27], we may restrict to the 2-connected case henceforth. Second, we consider a 2-connected G with a fixed planar embedding \mathcal{E} and use the GENERALIZED ANTIFACTOR-problem to test whether G admits a 3-augmentation $H \supseteq G$ with a planar embedding whose restriction to G equals \mathcal{E} . Finally, for a 2-connected G with variable embedding, we use an SPQR-tree of G to efficiently go through the possible planar embeddings of G with a dynamic program and to identify one such embedding that allows for a 3-augmentation, or conclude that no such exists.

Outline. After discussing related work below, we give necessary definitions in Section 1.1, including the GENERALIZED ANTIFACTOR-problem and SPQR-trees. In Section 2 we develop our algorithm for the 3-AUGMENTATION-problem, where we reduce to the 2-connected case in Section 2.1, and handle the fixed embedding in Section 2.2, and variable embedding in Section 2.3. Finally, in Section 3 we complete the loop back to the 3-EDGE COLORABILITY-problem for planar graphs. Lemmas marked with * are proven in the full version [13].

Related work. Hartmann, Rollin and Rutter [16] studied a similar augmentation problem for planar graphs, where we are only allowed to add edges (but no vertices) to the graph. In particular, for given $c, k \in \{1, \dots, 5\}$ they define the c -CONNECTED PLANAR k -REGULAR AUGMENTATION-problem where one seeks to add edges to a given planar graph G , so that the resulting supergraph H of G is planar, c -connected, and k -regular. Observe that the 2-CONNECTED PLANAR 3-REGULAR AUGMENTATION-problem is more restrictive than the 3-AUGMENTATION-problem: The former forbids to add new vertices, therefore refuses all input graphs with an odd number of vertices, and requires the result to be connected, therefore refusing all input graphs that are 3-regular and disconnected. In fact, reducing from PLANAR 3SAT, they show that 2-CONNECTED PLANAR 3-REGULAR AUGMENTATION is NP-complete [16, Theorem 3], while we show that 3-AUGMENTATION lies in P.

Let us mention a few more examples from the rich and diverse area of augmentation problems. Eswaran and Tarjan [12] pioneered the systematic investigation of augmentation problems. They presented algorithms to find in $\mathcal{O}(|V| + |E|)$ a smallest number of edges whose addition to a given (not necessarily planar) graph $G = (V, E)$ results in a 2-connected respectively 2-edge-connected graph (a connected graph with no bridge), while the weighted versions of either problem is NP-complete. If we additionally require the result to be planar, already both unweighted problems are NP-complete [18, 22]. Other problems of augmenting to a planar graph consider augmenting to a grid graph [3], or triangulating while minimizing the maximum degree [9, 19], avoiding separating triangles [4], creating a Hamiltonian cycle [11], or resulting in a chordal graph [20], just to name a few.

1.1 Preliminaries

All graphs considered here are finite, undirected, and contain no loops but possibly multiedges. We write $\|G\|$ for the *size* of G (its number of edges) and denote the degree of a vertex v by $\deg(v)$, the minimum degree in G by $\delta(G)$, and the maximum degree in G by $\Delta(G)$. A graph G is d -regular, for some non-negative integer d , if we have $\delta(G) = \Delta(G) = d$. A 3-regular graph is also called *cubic*, while a graph G is *subcubic* if $\Delta(G) \leq 3$.

A *bridge* in a graph G is an edge e whose removal increases the number of connected components, i.e., $G - e$ has strictly more components than G . Equivalently, e is a bridge if e is not contained in any cycle of G . A *bridgeless* graph is one that contains no bridge. Note that a bridgeless graph may be disconnected. On the other hand, for a positive integer k , a graph $G = (V, E)$ is k -connected if $|V| \geq k + 1$ and for any set U of $k - 1$ vertices in G the graph $G - U$ is connected. In particular, a graph G of maximum degree $\Delta(G) \leq 3$ is 2-connected if and only if G is connected and bridgeless. A 2-connected graph is sometimes also called *biconnected*, while a 3-connected graph is sometimes also called *triconnected*.

A *planar embedding* \mathcal{E} of a (planar) graph G is (in a sense that we need not make precise here) an equivalence class of crossing-free drawings of G in the plane. In particular, a planar embedding determines the set F of all *faces*, the distinguished *outer face* $f_0 \in F$, the clockwise ordering of incident edges around each vertex and the *boundary* of each face as a set of *facial walks*, each being a clockwise ordering of vertices and edges (with repetitions allowed). The edges and vertices incident to the outer face are called *outer edges* and *outer vertices*, while all others are *inner edges* and *inner vertices*. For every embedding \mathcal{E} of G we define the *flipped embedding* \mathcal{E}' to be the embedding obtained from \mathcal{E} by reversing the clockwise order of incident edges at each vertex. This operation changes neither the set of faces nor the outer face. Whitney's Theorem [30] states that a 3-connected planar graph G has a unique embedding (up to the choice of the outer face and flipping).

Generalized (Anti)factors. If G is a subgraph of H , denoted $G \subseteq H$, and v is a vertex of G , then we denote the degree of v in G by $\deg_G(v)$. If $V(G) = V(H)$, then G is called a *spanning* subgraph of H . If each vertex v of H is assigned a set $B(v) \subseteq \{0, \dots, \deg_H(v)\}$, then a spanning subgraph G of H is called a *B-factor* of H if and only if $\deg_G(v) \in B(v)$ for every vertex v . Lovász [21] introduced *B-factors* and the GENERALIZED FACTOR-problem that, given graph H and for each vertex v in H a set $B(v)$, asks whether H admits some *B-factor*. A set $B(v)$ is said to have a *gap of length* $\ell \geq 1$ if there is an integer $i \in B(v)$ such that $i + 1, \dots, i + \ell \notin B(v)$, and $i + \ell + 1 \in B(v)$. While the GENERALIZED FACTOR-problem is NP-complete in general [21], it can be solved in polynomial time if all gaps of each $B(v)$ have length one [8].

Now let $\bar{B}(v) \subseteq \{0, \dots, \deg_H(v)\}$ be another set assigned to each vertex v . A spanning subgraph G of H is called a *\bar{B} -antifactor*, if and only if $\deg_G(v) \notin \bar{B}(v)$. One can think of $\bar{B}(v)$ as forbidden degrees for v in G . The GENERALIZED ANTIFACTOR-problem asks whether H admits a *\bar{B} -antifactor*. Note that the set $\{0, \dots, \deg_H(v)\} \setminus \bar{B}(v)$ is finite, so the GENERALIZED ANTIFACTOR-problem is indeed a special case of the GENERALIZED FACTOR-problem. Therefore, an instance of the GENERALIZED ANTIFACTOR-problem with no two consecutive integers in any $\bar{B}(v)$ corresponds to an instance of the GENERALIZED FACTOR-problem with gaps of length at most one¹ and can be solved in polynomial time [8].

In Section 2.2 we use a theorem by Sebö [24], giving an efficient algorithm to compute generalized antifactors without two consecutive forbidden degrees.

► **Theorem 3** (Sebö [24]). *Let $H = (V, E)$ be a graph and for each vertex $v \in V$ let $\bar{B}(v) \subseteq \{0, \dots, \deg_H(v)\}$ be a set containing no two consecutive integers. Then we can compute a \bar{B} -antifactor in time $\mathcal{O}(|V| \cdot |E|)$, or conclude that no such exists.*

SPQR-Tree. The *SPQR-tree* is a tree-like data structure that compactly encodes all planar embeddings of a biconnected planar graph. It was introduced by Di Battista and Tamassia [10] and can be computed in linear time [15]. Its precise definition includes quite a number of technical terms, of which we define the crucial ones below. This makes our exposition self-contained, while also ensuring the established terminology for experienced readers. We give an illustrating example in Figure 2.

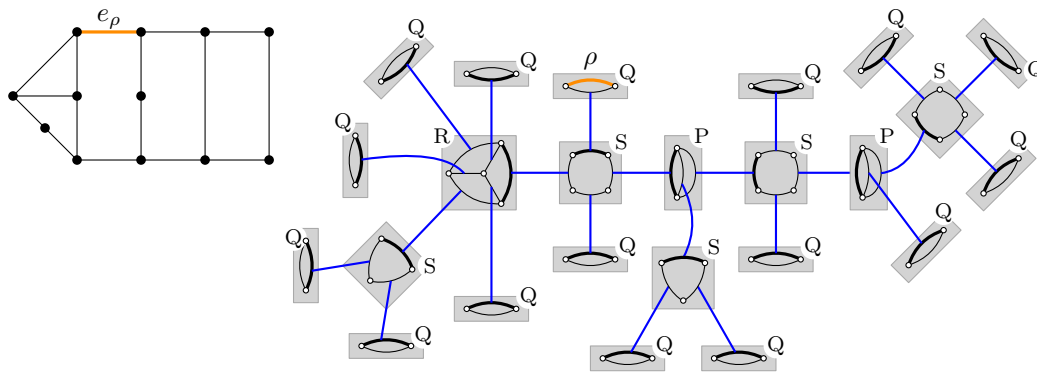
The SPQR-tree of a biconnected planar graph G is a rooted tree T , where each vertex μ of T is associated to a multigraph $\text{skel}(\mu)$ that is called the *skeleton* of μ . This multigraph $\text{skel}(\mu)$ must be of one of four types determining whether μ is an S-, a P-, a Q- or an R-vertex:

- S-vertex: $\text{skel}(\mu)$ is a simple cycle.
- P-vertex: $\text{skel}(\mu)$ consists of two vertices and at least three parallel edges.
- Q-vertex: $\text{skel}(\mu)$ consists of two vertices with two parallel edges.
- R-vertex: $\text{skel}(\mu)$ is triconnected.

Some of the edges of the skeletons can be marked as *virtual* edges. An edge $e = \mu\nu$ of the SPQR-tree T corresponds to two virtual edges, exactly one in $\text{skel}(\mu)$ and one in $\text{skel}(\nu)$. Conversely, each virtual edge corresponds to exactly one tree edge of T in this way. We refer again to Figure 2 for an example.

Under above conditions, the defining property of the SPQR-tree T is that G can be obtained by *gluing* along the virtual edges: For each tree edge $e = \mu\nu$, the skeletons $\text{skel}(\mu)$ and $\text{skel}(\nu)$ are identified at the corresponding endpoints of the two virtual edges associated to e and then the virtual edges are removed.

¹ Let us point out a subtlety here illustrating that this correspondence is not one-to-one. Requiring that $\bar{B}(v)$ does not contain two consecutive integers is stronger than requiring gaps of length 1 in $B(v) := \{0, \dots, \deg_H(v)\} \setminus \bar{B}(v)$. For example, consider a vertex v with $\deg_H(v) = 5$ and $\bar{B}(v) = \{1, 3, 4, 5\}$. Then $B(v) = \{0, 2\}$ has a gap of length 1, even though $\bar{B}(v)$ contained consecutive integers.



■ **Figure 2** A graph with an edge e_ρ (left) and its SPQR-tree rooted at the Q-vertex ρ corresponding to e_ρ (right). Each tree node μ shows the skeleton $\text{skel}(\mu)$ in which the virtual edge to its parent is shown thicker. The (blue) tree edges indicate the associated pairs of virtual edges.

We additionally require that no two S-vertices and no two P-vertices are adjacent in T , as otherwise the skeletons of two such vertices can be merged into the skeleton of a new vertex of the same type. Further, exactly one of the two parallel edges in a Q-vertex is a virtual edge while S-, P- and R-vertices contain only virtual edges. Under these conditions the SPQR-tree of G is unique. There is exactly one Q-vertex per edge in G and these form the leaves of the SPQR-tree. The inner S-, P- and R-vertices correspond more or less² to the separation pairs (that is, pairs of vertices forming a cut set) of G [10].

Assume that an arbitrary vertex ρ of T is fixed as the root. For some vertex μ in T let π be its parent. Further, let u, v be the endpoints of the virtual edge in $\text{skel}(\mu)$ associated with the tree edge $\mu\pi$ in T . Then the graph obtained by gluing $\text{skel}(\mu)$ with all skeletons in its subtree and without the virtual edge uv is called the *pertinent graph* of μ and denoted by $\text{pert}(\mu)$. Note that $\text{pert}(\mu)$ is always connected.

SPQR-Tree and Planar Embeddings. If the SPQR-tree T is rooted at a Q-vertex ρ corresponding to an edge e_ρ of G , then T represents all planar embeddings of G in which e_ρ is an outer edge [10]. When G is constructed by gluing corresponding virtual edges, one has the following choices on the planar embedding:

- Whenever the corresponding virtual edges of an S-, P- or R-vertex μ and its parent are glued together, this leaves two choices for the planar embedding: Having decided for an embedding \mathcal{E}_μ of $\text{pert}(\mu)$ already, we can insert \mathcal{E}_μ or the flipped embedding \mathcal{E}'_μ .
- The parallel virtual edges of a P-vertex μ associated to virtual edges of children can be permuted arbitrarily. Every permutation leads to a different planar embedding of $\text{skel}(\mu)$.
- Gluing at the virtual edge of a Q-vertex μ replaces the virtual edge uv by the “real” edge uv in G . This has no effect on the embedding.

Let \mathcal{E} be a planar embedding of G having e_ρ as an outer edge. Further, let μ be an inner vertex of the SPQR-tree and u_μ, v_μ be the endpoints of the virtual edge in $\text{skel}(\mu)$ corresponding to the parent edge of μ in T . Lastly, let \mathcal{E}_μ be the restriction of \mathcal{E} to $\text{pert}(\mu)$ and let f_μ^o be the outer face of \mathcal{E}_μ . As e_ρ is an outer edge of \mathcal{E} , it follows that u_μ and v_μ are outer vertices in \mathcal{E}_μ . The $u_\mu v_\mu$ -path in $\text{pert}(\mu)$ having f_μ^o to its left (right) is the *left (right) outer path* of \mathcal{E}_μ . Lastly, we define the *left (right) outer face* of \mathcal{E}_μ inside \mathcal{E} to be the face of \mathcal{E} left (right) of the left (right) outer path of \mathcal{E}_μ .

² In fact they correspond to so-called *split pairs*. However, we omit their formal discussion, as it is not needed here.

2 The 3-Augmentation-Problem

2.1 Reduction to the 2-Connected Case

► **Proposition 4.** *For a disconnected graph G with connected components G_1, \dots, G_k , $k \geq 2$, we have that*

- (i) *G has a 3-augmentation if and only if each G_i has a 3-augmentation, $i = 1, \dots, k$, and*
- (ii) *G has a 2-connected 3-augmentation if and only if each G_i has a 3-augmentation and no G_i is 3-regular, $i = 1, \dots, k$.*

Proof.

- (i) Any 3-augmentation of G is also a 3-augmentation of each G_i , showing already necessity. For sufficiency, observe that the 3-augmentations of different G_i are vertex-disjoint and hence their union is a 3-augmentation of G .
- (ii) Like above, a 2-connected 3-augmentation H of G is also a 3-augmentation of each G_i , $i = 1, \dots, k$. Moreover, as H is connected, each G_i has a vertex with at least one incident edge in $E(H) - E(G)$, showing that G_i is not 3-regular.

On the other hand, for $i = 1, \dots, k$ let H_i be a 3-augmentation of G_i . Without loss of generality each H_i is connected (hence 2-connected since 3-augmentations are bridgeless). As G_i is not 3-regular, we can pick an edge e_i from $E(H_i) - E(G_i)$, $i = 1, \dots, k$. Next, choose a planar embedding \mathcal{E} of the disjoint union $H_1 \cup \dots \cup H_k$ where each of e_1, \dots, e_k is an outer edge. Finally, add a copy of $K_4^{(2k)}$ into the outer face of \mathcal{E} , delete e_1, \dots, e_k , and connect the $2k$ degree-2 vertices of $H_1 \cup \dots \cup H_k$ with the $2k$ degree-2 vertices of $K_4^{(2k)}$ by a non-crossing matching. The result is a 2-connected 3-augmentation of G (by definition a 3-augmentation is bridgeless, so connectivity implies 2-connectivity). ◀

► **Proposition 5.** *A graph G admits a 3-augmentation if and only if $\Delta(G) \leq 3$ and each block of G admits a 3-augmentation.*

Proof. If G is bridgeless, then each connected component is a single block and thus admits a 3-augmentation by assumption. The disjoint union of these is a 3-augmentation of G .

Otherwise, consider G with a bridge $e = uv$. Let G_1 be the connected component of $G - e$ containing u , and let the remaining graph be $G_2 = G - G_1$. It is enough to show that if G_1 and G_2 have 3-augmentations H_1 respectively H_2 , then G has a 3-augmentation, too. To this end, consider an edge $e_1 \in E(H_1) - E(G_1)$ incident to u and an edge $e_2 \in E(H_2) - E(G_2)$ incident to v . These edges exist as $\deg_{G_1}(u), \deg_{G_2}(v) \leq \Delta(G) - 1 \leq 2$ but $\deg_{H_1}(u) = \deg_{H_2}(v) = 3$. Choose a planar embedding of $H_1 \cup H_2$ with e_1 and e_2 being outer edges. Denoting by a, b the endpoints of e_1, e_2 different from u, v , we see that $(H_1 - e_1) \cup (H_2 - e_2) \cup \{uv, ab\}$ is a 3-augmentation of G , as desired. ◀

2.2 The Fixed Embedding Setting

As usual for embedding-dependent problems for planar graphs, it makes sense to distinguish between the planar graph G being given with a fixed embedding that shall not be altered, and the setting with variable embedding where we solely have G as the input and shall find a suitable embedding for G or decide that no such exists. The 3-augmentation problem is formulated in the variable embedding setting. However, let us treat the variant with a fixed embedding first, as this will be a crucial subroutine for the variable embedding setting later.

► **Proposition 6.** *Let G be an n -vertex 2-connected planar multigraph of maximum degree $\Delta(G) \leq 3$ with a fixed planar embedding \mathcal{E} . Then we can compute in time $\mathcal{O}(n^2)$ a 3-augmentation H of G with a planar embedding \mathcal{E}_H whose restriction to G equals \mathcal{E} , or conclude that no such exists.*

Proof. Let V' denote the subset of the vertices of G with $\deg_G(v) \leq 2$ and let F denote the set of faces of \mathcal{E} . Note that since G is 2-connected, all $v \in V'$ have $\deg_G(v) = 2$. We consider the bipartite vertex-face incidence graph $I = (V' \cup F, E(I))$ with vertex-set $V' \cup F$ and edge-set $E(I) := \{vf \mid v \in V', f \in F, v \text{ is incident to } f\}$. Note that I has $O(n)$ vertices and at most $2n$ edges, since $\Delta(G) \leq 3$. We define an instance of the GENERALIZED ANTIFACTOR-problem by assigning each vertex x of I (corresponding to a vertex in G or a face in F) a set $\overline{B}(x) \subseteq \{0, \dots, \deg_I(x)\}$:

$$\overline{B}(x) := \begin{cases} \{0, 2\} & \text{for } x \in V' \\ \{1\} & \text{for } x \in F \end{cases}$$

Note that no $\overline{B}(x)$ contains two consecutive integers.

▷ **Claim 7.** Graph G admits a 3-augmentation H extending the embedding \mathcal{E} if and only if I admits a \overline{B} -antifactor.

Proof. First assume H is a 3-augmentation of G with a planar embedding \mathcal{E}_H that extends \mathcal{E} . Hence every edge $e \in E(H) - E(G)$ lies in a unique face of \mathcal{E} . We construct a \overline{B} -antifactor of I as follows. For each degree-2 vertex v of G , let f_v be the face of \mathcal{E} that contains the unique edge in $E(H) - E(G)$ incident to v . We claim that $J = (V' \cup F, \{vf_v \mid v \in V'\})$ is a \overline{B} -antifactor of I . In fact, $\deg_J(v) = 1$ for each degree-2 vertex $v \in V'$. Now if we would have $\deg_J(f) = 1$ for some face $f \in F$, then exactly one vertex $v \in V'$ has exactly one incident edge e lying in face f . In particular, the other endpoint of e is not a vertex of G . But then e is a bridge and H is not a 3-augmentation. Hence $\deg_J(f) \neq 1$ for each $f \in F$ and I indeed admits a \overline{B} -antifactor.

Conversely assume now that I has some \overline{B} -antifactor J . Then we construct the desired 3-augmentation H of G as follows. Inside each face f of \mathcal{E} with $\deg_J(f) > 0$ place a copy K_f of $K_4^{(\deg_J(f))}$. Connect the $\deg_J(f)$ degree-2 vertices $v \in V'$ with $vf \in E(J)$ by a non-crossing matching with the $\deg_J(f)$ degree-2 vertices of K_f . Call the resulting graph H and its resulting planar embedding \mathcal{E}_H . Then H is 2-connected (in particular bridgeless) as G is 2-connected and $\deg_J(f) \neq 1$ for each $f \in F$. Moreover, H is 3-regular. In fact, for each vertex $v \in V'$ we have $\deg_H(v) = \deg_G(v) + 1 = 3$, as J is a \overline{B} -antifactor. Finally, restricting \mathcal{E}_H to G gives back embedding \mathcal{E} . ◀

Now Claim 7 immediately finishes the proof because no $\overline{B}(x)$ contains two consecutive integers. Hence, by Sebö's algorithm [24] (cf. Theorem 3) we can compute a \overline{B} -antifactor of I in $\mathcal{O}(n^2)$ time, or conclude that no such exists. ◀

2.3 The Variable Embedding Setting

Even an unlabeled 2-connected subcubic planar graph G can have exponentially many different planar embeddings (e.g., the $(2 \times n)$ -grid graph). Thus, iterating over all embeddings of G and applying the algorithm from Proposition 6 to each of them is not a polynomial-time algorithm and hence no feasible approach for us. In this section we describe how to use the SPQR-tree of G to efficiently find a planar embedding \mathcal{E} of G such that there is a 3-augmentation H of G extending \mathcal{E} , or conclude that no such embedding exists. The algorithm from Proposition 6 will be an important subroutine.

► **Proposition 8.** *Let G be an n -vertex 2-connected planar graph of maximum degree $\Delta(G) \leq 3$. Then we can compute in $\mathcal{O}(n^2)$ time a 3-augmentation H of G or conclude that no such exists.*

Overview. The proof of Proposition 8 uses a bottom-up dynamic programming approach on the SPQR-tree T of G rooted at a Q-vertex ρ corresponding to some edge e_ρ in G . Consider a vertex $\mu \neq \rho$ in T . Let uv be the virtual edge in $\text{skel}(\mu)$ that is associated to the parent edge of μ . Recall that each embedding \mathcal{E} of G with e_ρ on the outer face, when restricted to the pertinent graph $\text{pert}(\mu)$, gives an embedding \mathcal{E}_μ of $\text{pert}(\mu)$ whose inner faces are also inner faces of \mathcal{E} , and with u and v being outer vertices of \mathcal{E}_μ . The outer face of \mathcal{E}_μ is composed of two (not necessarily edge-disjoint) u - v -paths; the left and right outer path of \mathcal{E}_μ , which are contained in the left and right outer face of \mathcal{E}_μ inside \mathcal{E} , respectively. We seek to partition the (possibly exponentially many) planar embeddings of $\text{pert}(\mu)$ with u, v on its outer face into a constant number of equivalence classes based on how many edges in a 3-augmentation of G could possibly “connect” $\text{pert}(\mu)$ with the rest of the graph G inside the left or right outer face of \mathcal{E}_μ inside \mathcal{E} . This corresponds³ to the number of degree-2 vertices on the left and right side in so-called *inner augmentations* of \mathcal{E}_μ . Loosely speaking, it will be enough for us to distinguish three cases for the left side (0, 1, or at least 2 connections), the symmetric three cases for the right side, and to record which of the nine resulting combinations are possible. Note that this grouping of embeddings of \mathcal{E}_μ into constantly many classes is the key insight that allows an efficient dynamic program.

Whether a particular equivalence class is realizable by some planar embedding \mathcal{E}_μ of $\text{pert}(\mu)$ will depend on the vertex type of μ (S-, P- or R-vertex) and the realizable equivalence classes of its children μ_1, \dots, μ_k . In the end, we shall conclude that the whole graph G has a 3-augmentation if and only if for the unique child μ of the root ρ of T the equivalence class of embeddings of $\text{pert}(\mu)$ for which neither the left nor the right side has any connections is non-empty.

Most of our arguments are independent of SPQR-trees and we instead consider so-called uv -graphs, which are slightly more general than pertinent graphs. We shall introduce inner augmentations of uv -graphs, which then give rise to label sets for uv -graphs, both in a fixed and variable embedding setting. These label sets encode the aforementioned number of connections between the uv -graph as a subgraph of G and the rest of G in a potential 3-augmentation. After showing that we can compute even variable label sets by resorting to the fixed embedding case and Proposition 6, we then present the final dynamic program along the rooted SPQR-tree T of G .

uv -Graphs and Labels. A uv -graph is a connected multigraph G_{uv} with $\Delta(G_{uv}) \leq 3$, two distinguished vertices u, v of degree at most 2, together with a planar embedding \mathcal{E}_{uv} such that u and v are outer vertices. A connected multigraph $H_{uv} \supseteq G_{uv}$ with planar embedding \mathcal{E}_H is an *inner augmentation* of G_{uv} if

- \mathcal{E}_H extends \mathcal{E}_{uv} and has u, v on its outer face,
- each of u, v has the same degree in H_{uv} as in G_{uv} ,
- every vertex of H_{uv} except for u, v has degree 1 or 3,
- every degree-1 vertex of H_{uv} lies in the outer face of \mathcal{E}_H and
- every bridge of H_{uv} that is not a bridge of G_{uv} is incident to a degree-1 vertex.

Because u, v are outer vertices in \mathcal{E}_H , one could add another edge e_{uv} (oriented from u to v) into the outer face of \mathcal{E}_H preserving planarity (this edge is not part of the inner augmentation). Then e_{uv} splits the outer face into two faces f_A, f_B left and right of e_{uv} , respectively. Each degree-1 vertex of H_{uv} now lies either inside f_A or f_B .

³ up to the fact that left and right outer path may share degree-2 vertices, each of which sends however its third edge into only one of the left and right outer face

We are interested in the number of degree-1 vertices in each of these faces of \mathcal{E}_H and write $d(H_{uv}, \mathcal{E}_H) = (a, b)$ if an inner augmentation H_{uv} of G_{uv} has exactly a degree-1 vertices inside f_A and exactly b degree-1 vertices inside f_B .

► **Lemma 9.** *Let H_{uv} be an inner augmentation of G_{uv} with $d(H_{uv}, \mathcal{E}_H) = (a, b)$. If $a \geq 2$, then G_{uv} has an inner augmentation H_{uv}^0 with $d(H_{uv}^0, \mathcal{E}_H^0) = (0, b)$ and an inner augmentation H_{uv}^1 with $d(H_{uv}^1, \mathcal{E}_H^1) = (1, b)$. A symmetric statement holds when $b \geq 2$.*

Proof. Add edge uv to the inner augmentation H_{uv} such that it has a degree-1 vertices in f_A . We add a copy of $K_4^{(a)}$ into f_A and identify the a degree-2 vertices of $K_4^{(a)}$ with the a degree-1 vertices in f_A in a non-crossing way. Ignoring edge uv , the obtained graph is the desired inner augmentation H_{uv}^0 with $d(H_{uv}^0, \mathcal{E}_H^0) = (0, b)$. We obtain H_{uv}^1 by additionally subdividing an edge of $K_4^{(a)}$ that is incident to f_A once and by attaching a degree-1 vertex to it into f_A . ◀

Motivated by Lemma 9, we focus on inner augmentations H_{uv} with $d(H_{uv}, \mathcal{E}_H) = (a, b)$ where $a, b \in \{0, 1\}$, and assign to H_{uv} in this case the *label* \mathbf{ab} with $\mathbf{a}, \mathbf{b} \in \{0, 1\}$.

The *embedded label set* $L_{\text{emb}}(G_{uv}, \mathcal{E}_{uv})$ contains all labels \mathbf{ab} such that there is an inner augmentation H_{uv} of G_{uv} with label \mathbf{ab} . Allowing other planar embeddings of G_{uv} , we further define the *variable label set* as $L_{\text{var}}(G_{uv}) = \bigcup_{\mathcal{E}} L_{\text{emb}}(G_{uv}, \mathcal{E})$, where \mathcal{E} runs over all planar embeddings of G_{uv} where u and v are outer vertices. As this in particular includes for each embedding \mathcal{E} of G_{uv} also the flipped embedding \mathcal{E}' of G_{uv} , it follows that $\mathbf{ab} \in L_{\text{var}}(G_{uv})$ if and only if $\mathbf{ba} \in L_{\text{var}}(G_{uv})$. Whenever this property holds for a (variable or embedded) label set, we call the label set *symmetric*. Hence, all variable label sets are symmetric, but embedded label sets may or may not be symmetric.

For brevity, let us use \star as a wildcard character, in the sense that if $\{\mathbf{x0}, \mathbf{x1}\}$ is in an embedded or variable label set for some $\mathbf{x} \in \{0, 1\}$, then we shorten the notation and replace them by a label $\mathbf{x}\star$. Symmetrically, we use the notation $\star\mathbf{x}$ and in particular define $\{\star\star\} := \{00, 01, 10, 11\}$. Using this notation, the eight possible symmetric label sets are:

$$\emptyset, \{00\}, \{01, 10\}, \{11\}, \{0\star, \star 0\}, \{00, 11\}, \{1\star, \star 1\}, \{\star\star\} \quad (1)$$

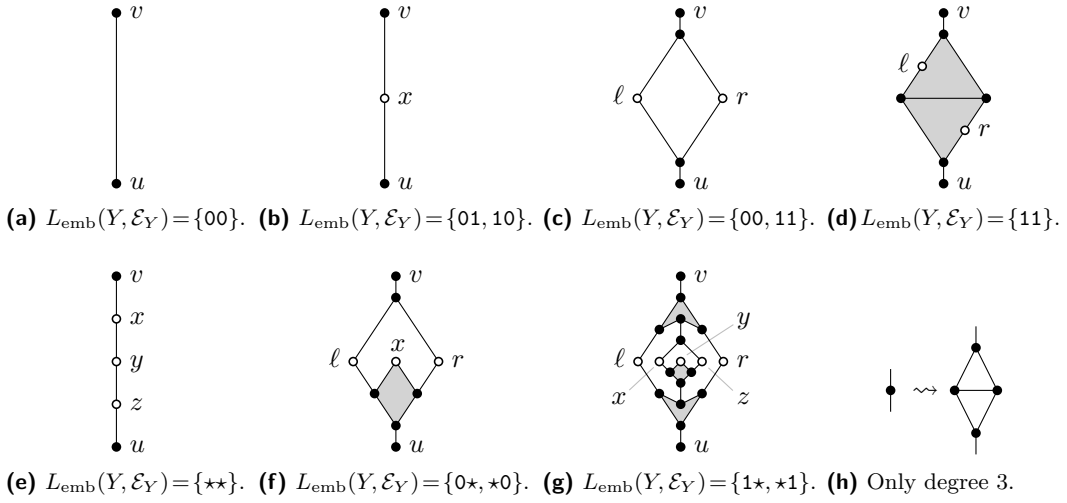
The following lemma reveals the significance of inner augmentations and label sets.

► **Lemma 10.** *Let G be a 2-connected graph with $\Delta(G) \leq 3$ and \mathcal{E} be an embedding of G with some outer edge $e = xy$. Further, let G_{uv} be the uv -graph obtained from G by deleting e and adding two new vertices u, v with edges ux and vy into the outer face of \mathcal{E} . Then G has a 3-augmentation if and only if $00 \in L_{\text{var}}(G_{uv})$.*

Proof. First let $H \supseteq G$ be a 3-augmentation of G and let \mathcal{E}_H be an embedding of H with $e = xy$ being an outer edge. Then deleting e and adding two new vertices u, v with edges ux and vy into the outer face of \mathcal{E}_H results in an inner augmentation H_{uv} of G_{uv} with respect to the embedding of G_{uv} inherited from \mathcal{E}_H . As adding an edge e_{uv} from u to v into H_{uv} gives a graph with no degree-1 vertices, we have $00 \in L_{\text{var}}(G_{uv})$.

Conversely, assume that $00 \in L_{\text{var}}(G_{uv})$. Then there is an embedding \mathcal{E}_{uv} of G_{uv} that allows for some inner augmentation H_{uv} with embedding \mathcal{E}_H for which $H_{uv} + e_{uv}$ has no degree-1 vertices, where $e_{uv} = uv$ denotes a new edge between u and v . Thus, in H_{uv} the vertices u and v have degree 1 (as in G_{uv}), every vertex of H_{uv} except u, v has degree 3, and the only bridges of H_{uv} are the edges ux and vy . Then we obtain a 3-augmentation H of G by removing ux, vy from H_{uv} and adding the edge xy into the outer face of \mathcal{E}_H . In case, H_{uv} already contains the edge xy , this is replaced by a copy of $K_4^{(2)}$ with two non-crossing edges between x, y and the two degree-2 vertices of $K_4^{(2)}$. ◀

62:10 Efficient Recognition of Subgraphs of Planar Cubic Bridgeless Graphs



■ **Figure 3** a–g The seven gadgets for the seven non-empty variable label sets $L_{\text{var}}(G_{uv})$ of a uv -graph G_{uv} . h Modification to locally replace a degree-2 vertex by four degree-3-vertices.

Gadgets. In our algorithm below, we aim to replace certain uv -graphs X (with variable embedding) by uv -graphs Y with fixed embedding \mathcal{E}_Y , such that the variable label set $L_{\text{var}}(X)$ equals the embedded label set $L_{\text{emb}}(Y, \mathcal{E}_Y)$. This will allow us to use Proposition 6 from the fixed embedding setting as a subroutine.

The following lemma describes seven uv -graphs, each with a fixed embedding, corresponding to the seven different non-empty variable label sets as given in (1). For this purpose, each such *gadget* is itself a uv -graph Y with a fixed embedding \mathcal{E}_Y .

► **Lemma* 11.** *For every uv -graph G_{uv} with $L_{\text{var}}(G_{uv}) \neq \emptyset$ there exists a gadget Y with an embedding \mathcal{E}_Y such that u, v are outer vertices and $L_{\text{emb}}(Y, \mathcal{E}_Y) = L_{\text{var}}(G_{uv})$.*

Lemma 11 is proven in the full version [13], but the claimed gadgets are shown in Figure 3.

Computing a Label Set. In our algorithm below we want to compute the variable label sets of $\text{pert}(\mu)$ for vertices μ of the rooted SPQR-tree T of G . As we will see, we can reduce this to a constant number of computations of embedded label sets of certain uv -graphs that are specifically crafted to encode all the possible embeddings of $\text{pert}(\mu)$. The following lemma describes how to do this.

► **Lemma* 12.** *Let G_{uv} be an n -vertex uv -graph and \mathcal{E}_{uv} a planar embedding where u and v are outer vertices. Then we can check each of the following in time $\mathcal{O}(n^2)$:*

- Whether $00 \in L_{\text{emb}}(G_{uv}, \mathcal{E}_{uv})$.
- Whether $01 \in L_{\text{emb}}(G_{uv}, \mathcal{E}_{uv})$ or $10 \in L_{\text{emb}}(G_{uv}, \mathcal{E}_{uv})$.
- Whether $11 \in L_{\text{emb}}(G_{uv}, \mathcal{E}_{uv})$.

In particular, if $L_{\text{emb}}(G_{uv}, \mathcal{E}_{uv})$ is symmetric, then this is sufficient to determine the exact embedded label set $L_{\text{emb}}(G_{uv}, \mathcal{E}_{uv})$.

The idea for Lemma 12 is similar to Lemma 10: For each check, we do some small local modifications to G_{uv} in order to obtain an embedded planar graph G_{uv}^+ that has a 3-augmentation extending its embedding if and only if $L_{\text{emb}}(G_{uv}, \mathcal{E}_{uv})$ contains the specific label. Then the result follows from Proposition 6. A full proof is given in the full version [13].

Algorithm for Variable Embedding. In order to decide whether a given biconnected planar graph G admits some planar embedding which admits a 3-augmentation, we use the SPQR-tree T of G . Rooting T at some Q-vertex ρ , the pertinent graph $\text{pert}(\mu)$ of a vertex μ in T is a subgraph of G . Moreover, if $u_\mu v_\mu$ is the virtual edge in $\text{skel}(\mu)$ associated to the parent edge of μ , then $\text{pert}(\mu)$ is a uv -graph (with u_μ, v_μ taking the roles of u, v in the uv -graph). Now the variable label set $L_{\text{var}}(\text{pert}(\mu))$ is a constant-size representation of all possible labels that any possible embedding of an inner augmentation of $\text{pert}(\mu)$ can have (having u_μ and v_μ on its outer face). The remainder of this section describes how the variable label sets of all vertices in the SPQR-tree can be computed by a bottom-up dynamic program.

► **Lemma* 13.** *Let μ be an inner R-, S-, or P-vertex of the SPQR-tree and μ_1, \dots, μ_k be its children. Further assume that the variable label sets $L_{\text{var}}(\text{pert}(\mu_i))$, for $i = 1, \dots, k$, are non-empty and known. Then the variable label set $L_{\text{var}}(\text{pert}(\mu))$ can be computed in time $\mathcal{O}(\|\text{skel}(\mu)\|^2)$.*

The idea to prove Lemma 13 is to consider $\text{skel}(\mu)$ with its essentially unique embedding and to replace for each μ_i the associated virtual edge by the embedded gadget Y from Lemma 11 (cf. Figure 3) with $L_{\text{emb}}(Y, \mathcal{E}_Y) = L_{\text{var}}(\text{pert}(\mu_i))$. Then the virtual edge uv associated to the parent of μ gets removed to obtain an embedded uv -graph on $\mathcal{O}(\|\text{skel}(\mu)\|)$ vertices, whose embedded label set can then be computed with Lemma 12. A full proof is given in the full version [13].

Lemma 13 computes the variable label set of an inner vertex of the SPQR-tree, requiring that the variable label sets of its children are non-empty. If this condition is not satisfied, i.e., at least one vertex μ has $L_{\text{var}}(\text{pert}(\mu)) = \emptyset$, then the following lemma applies:

► **Lemma 14.** *If $L_{\text{var}}(\text{pert}(\mu)) = \emptyset$ for some vertex μ of the SPQR-tree T of G , then G has no 3-augmentation.*

Proof. Assuming that G has a 3-augmentation H , we shall show that $L_{\text{var}}(\text{pert}(\mu)) \neq \emptyset$ for every vertex μ of T . If μ is the root, let u, v be the two unique vertices in $\text{skel}(\mu)$ (because $\mu = \rho$ is a Q-vertex). If μ is not the root, let u, v be the endpoints of the virtual edge associated to the parent edge of μ .

By the definition of labels, $L_{\text{var}}(\text{pert}(\mu)) \neq \emptyset$ if there is some inner augmentation of $\text{pert}(\mu)$ for at least one of its planar embeddings with u, v on its outer face. But the 3-augmentation H of G induces an inner augmentation of $\text{pert}(\mu)$ as follows: Let \mathcal{E}_H be a planar embedding of H with outer edge e_ρ and \mathcal{E}_G its restriction to G . Recall that then u, v are outer vertices of $\text{pert}(\mu)$ in \mathcal{E}_G . Consider the embedded subgraph of H consisting of $\text{pert}(\mu)$ and all vertices and edges of H inside inner faces of $\text{pert}(\mu)$ in \mathcal{E}_G . For each vertex $w \neq u, v$ on the outer face of $\text{pert}(\mu)$ in \mathcal{E}_G incident to an edge of H in the outer face of $\text{pert}(\mu)$, we add a new pendant edge at w into the outer face of $\text{pert}(\mu)$ in \mathcal{E}_G . The resulting graph is an inner augmentation of $\text{pert}(\mu)$ and hence $L_{\text{var}}(\text{pert}(\mu)) \neq \emptyset$. ◀

Now that we considered S-, P- and R-vertices, we are finally set up to prove Proposition 8. There we claim that we can decide in polynomial time whether a biconnected planar graph G with $\Delta(G) \leq 3$ has a 3-augmentation.

Proof of Proposition 8. As mentioned above, we use bottom-up dynamic programming on the SPQR-tree T of G rooted at an arbitrary Q-vertex ρ corresponding to an edge e_ρ in G .

The base cases are the leaves of T , all of which are Q-vertices. The variable label set of a leaf μ is $L_{\text{var}}(\text{pert}(\mu)) = \{00\}$: $\text{pert}(\mu)$ is just a single edge and the only inner augmentation of $\text{pert}(\mu)$ is $\text{pert}(\mu)$ itself, and as such has label 00.

Now let μ be an inner vertex of T and thus be either an S-, a P- or an R-vertex. All its children μ_1, \dots, μ_k have already been processed and their variable label sets $L_{\text{var}}(\text{pert}(\mu_i))$ are known. Then the variable label set $L_{\text{var}}(\text{pert}(\mu))$ can be computed in time $\mathcal{O}(\|\text{skel}(\mu)\|^2)$ (which is actually $\mathcal{O}(1)$ in case of a P-vertex) by Lemma 13. To apply this lemma, we need to guarantee that the variable label sets $L_{\text{var}}(\text{pert}(\mu_i))$ of the children are non-empty. If this is not the case, then by Lemma 14 graph G has no 3-augmentation and we can stop immediately.

It remains to consider the root ρ of the SPQR-tree. Recall that $\text{pert}(\rho) = G$. Following the setup of Lemma 10, let x, y be the two unique vertices of $\text{skel}(\rho)$ and xy be the unique non-virtual edge, i.e., the edge $e_\rho = xy$ of G . Let G_{uv} be the uv -graph obtained from $G = \text{pert}(\rho)$ by deleting $e_\rho = xy$ and adding two new pendant edges ux, vy . Note that x and y have the same degree in G_{uv} as in G . By Lemma 10, G has a 3-augmentation if and only if $00 \in L_{\text{var}}(G_{uv})$.

To check whether $00 \in L_{\text{var}}(G_{uv})$, let μ be the unique child of ρ . Thus we have $\text{pert}(\mu) = G - e_\rho$. We have already computed $L_{\text{var}}(\text{pert}(\mu))$ and can assume by Lemma 14 that it is non-empty. Consider the gadget Y with embedding \mathcal{E}_Y from Lemma 11 such that $L_{\text{emb}}(Y, \mathcal{E}_Y) = L_{\text{var}}(\text{pert}(\mu))$. Let u' and v' denote the two degree-1 vertices in Y . If both x and y have degree 3 in G (hence also in G_{uv}), then $L_{\text{var}}(G_{uv}) = L_{\text{var}}(\text{pert}(\mu)) = L_{\text{emb}}(Y, \mathcal{E}_Y)$ and we already know whether or not 00 is contained in these label sets.

If x has degree 2 in G (hence also degree 2 in G_{uv} , while degree 1 in $\text{pert}(\mu)$), then x receives a new edge in inner augmentations of G_{uv} but not in inner augmentations of $\text{pert}(\mu)$. For Y to model $L_{\text{var}}(G_{uv})$ instead of $L_{\text{var}}(\text{pert}(\mu))$, we subdivide in Y the edge at u' by a new vertex x' . Similarly, if y has degree 2 in G , we subdivide in Y the edge at v' . For the resulting graph Y' with embedding $\mathcal{E}_{Y'}$ it follows that $L_{\text{var}}(G_{uv}) = L_{\text{emb}}(Y', \mathcal{E}_{Y'})$ and we can check whether 00 is contained in these label sets by calling Lemma 12 on Y' with embedding $\mathcal{E}_{Y'}$. This takes constant time, as Y' has constant size.

The overall runtime is the time needed to construct the SPQR-tree plus the time spent processing each of its vertices. Gutwenger and Mutzel [15] show how to construct the SPQR-tree in time $\mathcal{O}(n)$. The time for the dynamic program traversing the SPQR-tree T is

$$\mathcal{O}\left(\sum_{\mu \in V(T)} \|\text{skel}(\mu)\|^2\right) \subseteq \mathcal{O}\left(\left(\sum_{\mu \in V(T)} \|\text{skel}(\mu)\|\right)^2\right) \subseteq \mathcal{O}(n^2),$$

where the first step uses that for a set of positive integers the sum of their squares is at most the square of their sum, and the second step uses that the SPQR-tree has linear size. \blacktriangleleft

3 Discussion and Open Problems

In this paper we showed how to test in polynomial time whether a planar graph G is a subgraph of some bridgeless cubic planar graph H . (We call such H a 3-augmentation of G .) Our motivation was to test whether G admits a proper 3-edge-coloring, because admitting a 3-augmentation is sufficient to conclude that $\chi'(G) \leq 3$. (This follows from the Four-Color-Theorem [1, 2] and the work of Tait [26].) However, there are 3-edge-colorable planar graphs with no 3-augmentation; $K_{2,3}$ is an easy example. For another class of examples, consider for instance any 3-connected 3-regular plane graph G (that is, the dual of a plane triangulation) and subdivide (with a new degree-2 vertex each) any set of at least two edges, where no two of these are incident to the same face of G (so their dual edges form a matching in the triangulation). The resulting graph G' has only one embedding (up to the choice of the outer face) and clearly no 3-augmentation. On the other hand, Conjecture 15 below predicts that G' is 3-edge colorable.

The computational complexity of the 3-EDGE COLORABILITY-problem for planar graphs remains open, while it is known to be NP-complete already for 3-regular, but not necessarily planar, graphs [17]. Similarly to our methods in Section 2.1, one can easily show that a planar subcubic graph is 3-edge-colorable if and only if all of its blocks (inclusion-maximal biconnected subgraphs) are 3-edge-colorable, i.e., 3-EDGE COLORABILITY reduces to the 2-connected case. A simple counting argument shows that a 2-connected subcubic graph G with exactly one degree-2 vertex is not 3-edge-colorable (independent of whether G is planar or not). The following conjecture, attributed to Grötzsch by Seymour [25], states that in the case of planar graphs, this is the only obstruction.

► **Conjecture 15** (Grötzsch, cf. [25]). *If G is a 2-connected planar graph of maximum degree $\Delta(G) \leq 3$, then G is 3-edge-colorable, unless it has exactly one vertex of degree 2.*

If Conjecture 15 is true, 3-EDGE COLORABILITY would be in P, as its condition is easy to check in linear time.

Finally, let us also briefly discuss planar graphs of maximum degree larger than 3. Vizing conjectured in 1965 that all planar graphs of maximum degree $\Delta \geq 6$ are Δ -edge-colorable, proving it only for $\Delta \geq 8$ [29]. As of today, it is known that all planar graphs of maximum degree $\Delta \geq 7$ are Δ -edge-colorable [14, 23, 31], and optimal edge colorings can be computed efficiently in these cases. The case $\Delta = 6$ is still open, while for $\Delta = 3, 4, 5$ there are planar graphs of maximum degree Δ that are not Δ -edge-colorable [29], and at least for $\Delta = 4, 5$ the Δ -EDGE COLORABILITY-problem is suspected to be NP-complete for planar graphs [6].

Generalizing Conjecture 15, Seymour's Exact Conjecture [25] states that every planar graph G is $\lceil \eta'(G) \rceil$ -edge-colorable, where $\eta'(G)$ denotes the fractional chromatic index of G . It is worth noting that Seymour's Exact Conjecture implies Vizing's Conjecture, as well as the Four-Color-Theorem; see e.g., the recent survey [5].

References

- 1 Kenneth Appel and Wolfgang Haken. Every planar map is four colorable. Part I: Discharging. *Illinois Journal of Mathematics*, 21(3):429–490, 1977. doi:10.1215/ijm/1256049011.
- 2 Kenneth Appel and Wolfgang Haken. Every planar map is four colorable. Part II: Reducibility. *Illinois Journal of Mathematics*, 21(3):491–567, 1977. doi:10.1215/ijm/1256049012.
- 3 Sandeep N. Bhatt and Stavros S. Cosmadakis. The Complexity of Minimizing Wire Lengths in VLSI Layouts. *Information Processing Letters*, 25(4):263–267, 1987. doi:10.1016/0020-0190(87)90173-6.
- 4 Therese Biedl, Goos Kant, and Michael Kaufmann. On Triangulating Planar Graphs under the Four-Connectivity Constraint. *Algorithmica*, 19(4):427–446, 1997. doi:10.1007/PL00009182.
- 5 Yan Cao, Guantao Chen, Guangming Jing, Michael Stiebitz, and Bjarne Toft. Graph Edge Coloring: A Survey. *Graphs and Combinatorics*, 35(1):33–66, 2019. doi:10.1007/s00373-018-1986-5.
- 6 Marek Chrobak and Takao Nishizeki. Improved Edge-Coloring Algorithms for Planar Graphs. *Journal of Algorithms*, 11(1):102–116, 1990. doi:10.1016/0196-6774(90)90032-A.
- 7 Richard Cole and Lukasz Kowalik. New Linear-Time Algorithms for Edge-Coloring Planar Graphs. *Algorithmica*, 50(3):351–368, 2008. doi:10.1007/s00453-007-9044-3.
- 8 Gérard Cornuéjols. General Factors of Graphs. *Journal of Combinatorial Theory, Series B*, 45(2):185–198, 1988. doi:10.1016/0095-8956(88)90068-8.
- 9 Hubert de Fraysseix and Patrice Ossona de Mendez. Regular Orientations, Arboricity, and Augmentation. In Roberto Tamassia and Ioannis G. Tollis, editors, *Graph Drawing. GD 1994*, volume 894 of *Lecture Notes in Computer Science*, pages 111–118, 1994. doi:10.1007/3-540-58950-3_362.
- 10 Giuseppe Di Battista and Roberto Tamassia. On-Line Planarity Testing. *SIAM Journal on Computing*, 25(5):956–997, 1996. doi:10.1137/S0097539794280736.

- 11 Emilio Di Giacomo and Giuseppe Liotta. The Hamiltonian Augmentation Problem and Its Applications to Graph Drawing. In Md. Saidur Rahman and Satoshi Fujita, editors, *WALCOM: Algorithms and Computation. WALCOM 2010*, volume 5942 of *Lecture Notes in Computer Science*, pages 35–46, 2010. doi:10.1007/978-3-642-11440-3_4.
- 12 Kapali P. Eswaran and R. Endre Tarjan. Augmentation Problems. *SIAM Journal on Computing*, 5(4):653–665, 1976. doi:10.1137/0205044.
- 13 Miriam Goetze, Paul Jungeblut, and Torsten Ueckerdt. Efficient Recognition of Subgraphs of Planar Cubic Bridgeless Graphs. arXiv preprint, 2022. arXiv:2204.11750.
- 14 Stefan Grünewald. *Chromatic Index Critical Graphs and Multigraphs*. PhD thesis, Universität Bielefeld, 2000. URL: <https://pub.uni-bielefeld.de/record/2303675>.
- 15 Carsten Gutwenger and Petra Mutzel. A Linear Time Implementation of SPQR-Trees. In Joe Marks, editor, *Graph Drawing. GD 2000*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90, 2001. doi:10.1007/3-540-44541-2_8.
- 16 Tanja Hartmann, Jonathan Rollin, and Ignaz Rutter. Regular augmentation of planar graphs. *Algorithmica*, 73(2):306–370, 2015. doi:10.1007/s00453-014-9922-4.
- 17 Ian Holyer. The NP-Completeness of Edge-Coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981. doi:10.1137/0210055.
- 18 Goos Kant and Hans L. Bodlaender. Planar Graph Augmentation Problems. In Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures. WADS 1991*, volume 519 of *Lecture Notes in Computer Science*, pages 286–298, 1991. doi:10.1007/BFb0028270.
- 19 Goos Kant and Hans L. Bodlaender. Triangulating Planar Graphs While Minimizing the Maximum Degree. *Information and Computation*, 135(1):1–14, 1997. doi:10.1006/inco.1997.2635.
- 20 Jan Kratochvíl and Michal Vaner. A note on planar partial 3-trees. arXiv preprint, 2012. arXiv:1210.8113.
- 21 László Lovász. The Factorization of Graphs. II. *Acta Mathematica Academiae Scientiarum Hungarica*, 23(1–2):223–246, 1972. doi:10.1007/BF01889919.
- 22 Ignaz Rutter and Alexander Wolff. Augmenting the Connectivity of Planar and Geometric Graphs. *Electronic Notes in Discrete Mathematics*, 31:53–56, 2008. doi:10.1016/j.endm.2008.06.009.
- 23 Daniel P. Sanders and Yue Zhao. Planar Graphs of Maximum Degree Seven are Class I. *Journal of Combinatorial Theory, Series B*, 83(2):201–212, 2001. doi:10.1006/jctb.2001.2047.
- 24 András Sebő. General Antifactors of Graphs. *Journal of Combinatorial Theory, Series B*, 58(2):174–184, 1993. doi:10.1006/jctb.1993.1035.
- 25 Paul D. Seymour. On Tutte’s Extension of the Four-Colour Problem. *Journal of Combinatorial Theory, Series B*, 31(1):82–94, 1981. doi:10.1016/S0095-8956(81)80013-5.
- 26 Peter G. Tait. 10. Remarks on the previous Communication. *Proceedings of the Royal Society of Edinburgh*, 10:729–731, 1880. doi:10.1017/S0370164600044643.
- 27 Robert E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.
- 28 Robert E. Tarjan. A Note on Finding the Bridges of a Graph. *Information Processing Letters*, 2(6):160–161, 1974. doi:10.1016/0020-0190(74)90003-9.
- 29 Vadim G. Vizing. Critical Graphs with Given Chromatic Class. *Akademiya Nauk SSSR. Sibirskoe Otdelenie. Institut Matematiki. Diskretnyĭ Analiz. Sbornik Trudov*, pages 9–17, 1965. In Russian.
- 30 Hassler Whitney. 2-Isomorphic Graphs. *American Journal of Mathematics*, 55(1):245–254, 1933. doi:10.2307/2371127.
- 31 Limin Zhang. Every Planar Graph with Maximum Degree 7 Is of Class 1. *Graphs and Combinatorics*, 16(4):467–495, 2000. doi:10.1007/s003730070009.

Adaptive-Adversary-Robust Algorithms via Small Copy Tree Embeddings

Bernhard Haepler ✉

Carnegie Mellon University, Pittsburgh, PA, USA
ETH Zürich, Switzerland

D. Ellis Hershkowitz ✉

Carnegie Mellon University, Pittsburgh, PA, USA

Goran Zuzic ✉

ETH Zürich, Switzerland

Abstract

Embeddings of graphs into distributions of trees that preserve distances in expectation are a cornerstone of many optimization algorithms. Unfortunately, online or dynamic algorithms which use these embeddings seem inherently randomized and ill-suited against adaptive adversaries.

In this paper we provide a new tree embedding which addresses these issues by *deterministically* embedding a graph into a single tree containing $O(\log n)$ copies of each vertex while preserving the connectivity structure of every subgraph and $O(\log^2 n)$ -approximating the cost of every subgraph. Using this embedding we obtain the first deterministic bicriteria approximation algorithm for the online covering Steiner problem as well as the first poly-log approximations for demand-robust Steiner forest, group Steiner tree and group Steiner forest.

2012 ACM Subject Classification Mathematics of computing → Trees; Mathematics of computing → Approximation algorithms; Mathematics of computing → Paths and connectivity problems; Theory of computation → Random projections and metric embeddings

Keywords and phrases Tree metrics, metric embeddings, approximation algorithms, group Steiner forest, group Steiner tree, demand-robust algorithms, online algorithms, deterministic algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.63

Related Version *Full Version:* <https://arxiv.org/pdf/2102.05168.pdf>

Funding Supported in part by NSF grants CCF-1527110, CCF-1618280, CCF-1814603, CCF-1910588, NSF CAREER award CCF-1750808, a Sloan Research Fellowship, funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (ERC grant agreement 949272) and Swiss National Foundation (project grant 200021-184735).

D. Ellis Hershkowitz: Supported by the Air Force Office of Scientific Research under award number FA9550-20-1-0080.

1 Introduction

Probabilistic embedding of general metrics into distributions over trees are one of the most versatile tools in combinatorial and network optimization. The beauty and utility of these tree embeddings comes from the fact that their application is often simple, yet extremely powerful. Indeed, when modeling a network with length, costs, or capacities as a weighted graph, these embeddings often allow one to pretend that the graph is a tree. A common template for countless network design algorithms is to (1) embed the input weighted graph G into a randomly sampled tree T that approximately preserves the weight structure of G ; (2) solve the input problem on T and; (3) project the solution on T back into G .



© Bernhard Haepler, D. Ellis Hershkowitz, and Goran Zuzic;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 63; pp. 63:1–63:14



Leibniz International Proceedings in Informatics

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

A long and celebrated line of work [32, 3, 6, 17] culminated in the embedding of Fakcharoenphol, Rao and Talwar [17] – henceforth the “FRT embedding” – which showed that any weighted graph on n nodes can be embedded into a distribution over weighted trees in a way that $O(\log n)$ -approximately preserves distances in expectation. Together with the above template this reduces many graph problems to much easier problems on trees at the cost of an $O(\log n)$ approximation factor. This has led to a myriad of approximation, online, and dynamic algorithms with poly-logarithmic approximations and competitive ratios for NP-hard problems such as for k -server [5], metrical task systems [8], group Steiner tree and group Steiner forest [2, 36, 21], buy-at-bulk network design [4] and (oblivious) routing [37]. For many of these problems tree embeddings are the only known way of obtaining such algorithms on general graphs.

However, probabilistic tree embeddings have one drawback: Algorithms based on them naturally require randomization and their approximation guarantees only hold in expectation. For approximation algorithms – i.e., in the offline setting – there are derandomization tools, such as the FRT derandomizations given in [12, 17, 7], to overcome these issues. These derandomization results are so general that essentially any offline algorithm based on tree embeddings can be transformed into a deterministic algorithm with matching approximation guarantees (with only a moderate increase in running time). Unfortunately, these strategies are not applicable to online or dynamic settings where an adversary progressively reveals the input. Indeed, most online and dynamic algorithms that use FRT are randomized (e.g. [23, 28, 2, 19, 8, 36, 14, 15]).

This overwhelming evidence in the literature is driven by a well-known and fundamental barrier to the use of probabilistic tree embeddings in deterministic online and dynamic algorithms. More specifically and even worse, this is a barrier which prevents these algorithms from working against all but the weakest type of adversary. In particular, designing an online or dynamic algorithm which is robust to an oblivious adversary (which fixes all requests in advance, independently of the algorithm’s randomness) is often much easier than designing an algorithm which is robust to an adaptive adversary (which chooses the next request based on the algorithm’s current solution). As the actions of a deterministic algorithm can be fully predicted this distinction only holds for randomized algorithms – any deterministic algorithm has to always work against an adaptive adversary. For these reasons, many online and dynamic algorithms have exponentially worse competitive ratios in the deterministic or adaptive adversary setting than in the oblivious adversary setting. This is independent of computational complexity considerations.

The above barrier results from a repeatedly recognized and seemingly unavoidable phenomenon which prevents online algorithms built on FRT from working against adaptive adversaries. Specifically, there are graphs where every tree embedding must have many node pairs with polynomially-stretched distances [6]. There is nothing that prevents an adversary then from learning through the online algorithm’s responses which tree was sampled and then tailoring the remainder of the online instance to pairs of nodes that have highly stretched distances. The exact same phenomenon occurs in the dynamic setting; see, for example, [23] and [28] for dynamic algorithms with expected cost guarantees that only hold against oblivious adversaries because they are based on FRT. In summary, online and dynamic algorithms that use probabilistic tree embeddings seem inherently randomized and seem to necessarily only work against adversaries oblivious to this randomness.

Similar, albeit not identical,¹ issues also arise in other settings, most notably demand-robust optimization. The demand-robust model is a well-studied model of optimization under uncertainty [13, 30, 18, 25, 26, 22] in which an algorithm first buys a partial solution given a large collection of potential problem instances. An “adaptive adversary” then chooses which of the potential instances must be solved and the algorithm must extend its partial solution to solve the selected instance at inflated costs. The adversary is adaptive in the sense that it chooses the final instance with full knowledge of the algorithm’s partial solution. To thwart an algorithm which reduces a demand-robust problem to its tree version via a sampled FRT tree, the adversary can present a collection of potential instances which for every tree T in the FRT distribution contains an instance for which T is an arbitrarily bad approximation and then always choose the worst-case problem instance. The fact that there do not exist any demand-robust algorithms which use FRT despite this setting having received considerable attention seems at least partially due to the issues pointed out here.

Overall it seems fair to say that prior to this work tree embeddings seemed fundamentally incapable of enabling adaptive-adversary-robust and deterministic algorithms in several well-studied settings.

1.1 Our Contributions

We provide a new type of metric embedding – the copy tree embedding – which is deterministic and therefore also adaptive-adversary-robust. Specifically, we show that any weighted graph G can be deterministically embedded into a single weighted tree with a small number of copies for each vertex. Any subgraph of G will project onto this tree in a connectivity and approximate-cost preserving way.

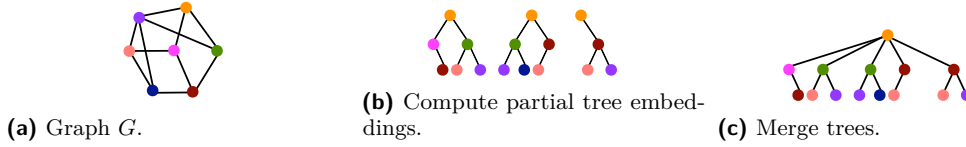
To precisely define our embeddings we define a copy mapping ϕ which maps a vertex v to its copies.

► **Definition 1** (Copy Mapping). *Given vertex sets V and V' we say $\phi : V \rightarrow 2^{V'}$ is a copy mapping if every node has at least one copy (i.e. $|\phi(v)| \geq 1$ for all $v \in V$), copies are disjoint (i.e. $\phi(v) \cap \phi(u) = \emptyset$ for $u \neq v$) and every node in V' is a copy of some node (i.e. for every $v' \in V'$ there is some $v \in V$ where $v' \in \phi(v$). For $v' \in V'$, we use the shorthand $\phi^{-1}(v')$ to stand for the unique $v \in V$ such that $v' \in \phi(v$).*

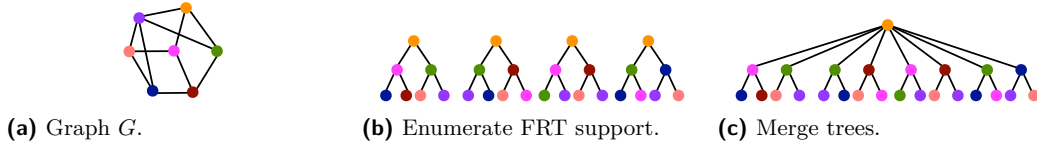
A copy tree embedding for a weighted graph G now simply consists of a tree T on copies of vertices of G with one distinguished root and two mappings $\pi_{G \rightarrow T}$ and $\pi_{T \rightarrow G}$ which map subsets of edges from G to T and from T to G in a way that preserves connectivity and approximately preserves costs. We say that two vertex subsets U, W are connected in a graph if there is a $u \in U$ and $w \in W$ such that u and w are connected. We also say that a mapping $\pi : 2^E \rightarrow 2^{E'}$ is *monotone* if for every $A \subseteq B$ we have that $\pi(A) \subseteq \pi(B)$. A rooted tree $T = (V, E, w)$ is *well-separated* if for all edges e if e' is a child edge of e in T then $w(e') \leq \frac{1}{2}w(e)$. In the below for $F \subseteq E$ we let $w(F) := \sum_{e \in F} w(e)$.

► **Definition 2** (α -Approximate Copy Tree Embedding with Copy Number χ). *Let $G = (V, E, w)$ be a weighted graph with some distinguished vertex $r \in V$ called the root. An α -approximate copy tree embedding with copy number χ consists of a weighted rooted tree $T = (V', E', w')$, a copy mapping $\phi : V \rightarrow 2^{V'}$ and edge mapping functions $\pi_{G \rightarrow T} : 2^E \rightarrow 2^{E'}$ and $\pi_{T \rightarrow G} : 2^{E'} \rightarrow 2^E$ where $\pi_{T \rightarrow G}$ is monotone and:*

¹ We remark that, unlike the online and dynamic setting, the barrier to obtaining demand-robust algorithms which work against the “adaptive adversary” implicit in the setting is merely computational and thus seems potentially less inherent.



■ **Figure 1** Illustration of our first construction where we merge $O(\log n)$ partial tree embeddings.



■ **Figure 2** Illustration of our second construction where we merge the $O(n \log n)$ trees in the FRT support.

1. **Connectivity Preservation:** For all $F \subseteq E$ and $u, v \in V$ if u, v are connected by F , then $\phi(u), \phi(v) \subseteq V'$ are connected by $\pi_{G \rightarrow T}(F)$. Symmetrically, for all $F' \subseteq E'$ and $u', v' \in V'$ if u' and v' are connected by F' then $\phi^{-1}(u')$ and $\phi^{-1}(v')$ are connected by $\pi_{T \rightarrow G}(F')$.
2. **α -Cost Preservation:** For any $F \subseteq E$ we have $w(F) \leq \alpha \cdot w'(\pi_{G \rightarrow T}(F))$ and for any $F' \subseteq E'$ we have $w'(F') \leq w(\pi_{T \rightarrow G}(F'))$.
3. **Copy Number:** $|\phi(v)| \leq \chi$ for all $v \in V$ and $\phi(r) = \{r'\}$ where r' is the root of T .

A copy tree embedding is efficient if T , ϕ , and $\pi_{T \rightarrow G}$ are deterministically poly-time computable and well-separated if T is well-separated.

We emphasize that, whereas standard tree embeddings guarantee costs are preserved in expectation, our copy tree embeddings preserve costs deterministically. Also notice that for efficient copy tree embeddings we do not require that $\pi_{G \rightarrow T}$ is efficiently computable; this is because $\pi_{G \rightarrow T}$ will be used in our analyses but not in any of our algorithms. The idea of embeddings which map vertices to several copies has previously been explored by [9] and was recently explored in a concurrent work of [20]. The key difference between these works and our own is that the number of copies that each vertex is mapped to is unboundedly large (in the case of [9]) or only small in expectation (in the case of [20]). On the other hand, the analogue of α -cost preservation in [9] (“path preservation”) is stronger than our α -cost preservation.

We first give two copy tree embedding constructions which trade off between the number of copies and cost preservation. Both constructions are based on the idea of merging appropriately chosen tree embeddings as pictured in Figure 1 and Figure 2 where we color nodes according to the node whose copy they are.

Construction 1: Merging Partial Tree Embeddings (full version). The cornerstone of our first construction is the idea of merging embeddings which give good *deterministic* distance preservation. If our goal is to embed the entire input metric into a tree this is impossible. However, it is possible to embed a random constant fraction of nodes in an input metric into a tree in a way that deterministically preserves distances of the embedded nodes; an embedding which we call a “partial tree embedding” (see also [24, 29]). We then use the method of conditional expectation to derandomize a node-weighted version of this random process and apply this derandomization $O(\log n)$ times, down-weighting nodes as

they are embedded. The result of this process is $O(\log n)$ partial tree embeddings where a multiplicative-weights-type argument shows that each node appears in a constant fraction of these embeddings. Merging these $O(\log n)$ embeddings gives our copy tree while an Euler-tour-type proof shows that subgraphs of the input graph can be mapped to our copy tree in a cost and connectivity-preserving fashion. The following theorem summarizes our first construction.

► **Theorem 3.** *There is a poly-time deterministic algorithm which given any weighted graph $G = (V, E, w)$ and root $r \in V$ computes an efficient and well-separated $O(\log^2 n)$ -approximate copy tree embedding with copy number $O(\log n)$.*

Construction 2: Merging FRT Support (full version). Our second construction follows from a known fact that the size of the support of the FRT distribution can be made $O(n \log n)$ and this support can be computed deterministically in poly-time [12]. Merging each tree in this support at the root and some simple probabilistic method arguments give a copy tree embedding that is $O(\log n)$ -cost preserving but with an $O(n \log n)$ copy number. Equivalently, it can be inferred from [9]. The next theorem summarizes this construction.

► **Theorem 4.** *There is a poly-time deterministic algorithm which given any weighted graph $G = (V, E, w)$ and root $r \in V$ computes an efficient and well-separated $O(\log n)$ -approximate copy tree embedding with copy number $O(n \log n)$.*

While our second construction achieves a slightly better cost bound than our first construction, it has the significant downside of a linear copy number. Notably, this linear copy number makes our second construction unsuitable for some applications, including, for example, our second application as described below. Moreover, our first construction also has several desirable properties which our second does not which we expect might be useful for future applications. These include: (1) $\pi_{G \rightarrow T}$ is monotone (in addition to $\pi_{T \rightarrow G}$ being monotone as stipulated by Definition 2); (2) if u and v are connected by $F \subseteq E$ then $\Omega(\log n)$ vertices of $\phi(u)$ are connected to $\Omega(\log n)$ vertices of $\phi(v)$ in $\pi_{G \rightarrow T}(F)$ (as opposed to just one vertex of $\phi(u)$ and one vertex of $\phi(v)$ as in Definition 2) and; (3) if u is connected to r by $F \subseteq E$ then every vertex in $\phi(u)$ is connected to $\phi(r)$ in $\pi_{G \rightarrow T}(F)$ (as opposed to just one vertex of $\phi(u)$ as in Definition 2).

We next apply our constructions to obtain new results for several online and demand-robust connectivity problems whose history we briefly summarize now. Group Steiner tree and group Steiner forest are two well-studied generalizations of set cover and Steiner tree. In the group Steiner tree problem, we are given a weighted graph $G = (V, E, w)$ and groups $g_1, \dots, g_k \subseteq V$ and must return a subgraph of G of minimum weight which contains at least one vertex from each group. The group Steiner forest problem generalizes group Steiner tree. Here, we are given $A_i, B_i \subseteq V$ pairs and for each i we must connect some vertex from A_i to some vertex in B_i . [2] and [36] each gave a poly-log approximation for online group Steiner tree and forest respectively but both of these approximation guarantees are randomized and only hold against oblivious adversaries because they rely on FRT. Indeed, [2] posed the existence of a deterministic poly-log approximation for online group Steiner tree as an open question which has since been restated several times [11, 10].

Another well-studied generalization of group Steiner tree is the covering Steiner problem which is defined as group Steiner tree but where we are additionally given a value $r_i \in (0, |g_i|]$ for each group g_i and must connected at least an r_i vertices of g_i in our subgraph. This problem was introduced by [35] and further studied in several follow-up works [16, 27].

Similarly, while demand-robust minimum spanning tree and special cases of demand-robust Steiner tree have received considerable attention [13, 34, 33], there are no known poly-log approximations for demand-robust Steiner tree, group Steiner tree or group Steiner forest.

The reason our embeddings are well-suited to group Steiner problems and its generalizations is that mapping it onto a copy tree embedding simply results in another instance of the group Steiner tree problem, this time on a tree. Indeed, our embeddings almost immediately reduce the open question of [2] – solving online group Steiner tree and forest deterministically on a general graph – to its tree case (see the full version for details). Equivalently, this reduction can be inferred from the embeddings [9], though [2] seems to have overlooked this connection.

Application 1: Deterministic Online Covering Steiner (Section 3). In our first application we make progress on the open question of [2] by showing that the online covering Steiner problem admits a *bicriteria* deterministic poly-log approximation. Specifically, note that the covering Steiner problem generalizes group Steiner tree but unlike group Steiner tree it admits a natural bicriteria relaxation: instead of connecting, for example, $\frac{1}{2}$ of the nodes in each group we could require that our algorithm only connects, say, $\frac{(1-\epsilon)}{2}$ of all nodes in each group for some $\epsilon > 0$. Thus, our result can be seen as showing that there is indeed a deterministic poly-log competitive algorithm for online group Steiner tree – as posed in the above open question of [2] – *provided the algorithm can be bicriteria* in the relevant sense. We use our embeddings for this application. Those of [9] or [20] are not suitable for our first application since this application requires a bound on the number of copies of each vertex. More formally, we obtain a deterministic poly-log bicriteria approximation for this problem which connects at least $\frac{1-\epsilon}{2}$ of the nodes in each group (notated “ $(1-\epsilon)$ -connection competitive” below) by using our copy tree embeddings and a “water-filling” algorithm to solve the tree case.

► **Theorem 5.** *There is a deterministic poly-time algorithm for online covering Steiner (on general graphs) which is $O(\frac{\log^3 n}{\epsilon} \cdot \max_i \frac{|g_i|}{r_i})$ -cost-competitive and $(1-\epsilon)$ -connection-competitive.*

As we later observe, providing a deterministic poly-log-competitive algorithm for the online covering Steiner problem with any constant bicriteria relaxation is strictly harder than providing a deterministic poly-log-competitive algorithm for online (non-group) Steiner tree. Thus, this result also generalizes the fact that a deterministic poly-log approximation is known for online (non-group) Steiner tree [31]. Additionally, as a corollary we obtain the first non-trivial deterministic approximation algorithm for online group Steiner tree – albeit one with a linear dependence on the maximum group size.²

► **Corollary 6.** *There is an $O(N \log^3 n)$ -competitive deterministic algorithm for online group Steiner tree where $N := \max_i |g_i|$ is the maximum group size.*

We next adapt and apply our embeddings in the demand-robust setting.

Application 2: Demand-Robust Steiner Problems (full version). We begin by generalizing copy tree embeddings to demand-robust copy tree embeddings. Roughly, these are copy tree embeddings which simultaneously work well for every possible demand-robust scenario.

² We explicitly note here that this bicriteria guarantee does not yield a solution to the open problem of [2] of finding a poly-log deterministic approximation to the online group Steiner tree problem.

We then adapt our analysis from our previous constructions to show that these copy tree embeddings exist. Lastly, we apply demand-robust copy tree embeddings to give poly-log approximations for the demand-robust versions of several Steiner problems – Steiner forest, group Steiner tree and group Steiner forest – for which, prior to this work, nearly nothing was known. In particular, the only non-trivial algorithms known for demand-robust Steiner problems prior to this work are an algorithm for demand-robust Steiner tree [13] and an algorithm for demand-robust Steiner forest *on trees* with exponential scenarios [18] (which is, in general, incomparable to the usual demand-robust setting). To show these results, we apply our demand-robust copy tree embeddings to reduce these problems to their tree case. Thus, we also give our results on trees which are themselves non-trivial.

► **Theorem 7.** *There is a randomized poly-time $O(\log^2 n)$ -approximation algorithm for the demand-robust group Steiner tree problem on weighted trees.*

► **Theorem 8.** *There is a randomized poly-time $O(D \cdot \log^3 n)$ -approximation algorithm for the demand-robust group Steiner forest problem on weighted trees of depth D .*

► **Theorem 9.** *There is a randomized poly-time $O(\log^4 n)$ -approximation algorithm for the demand-robust group Steiner tree problem on weighted graphs.*

► **Theorem 10.** *There is a randomized poly-time $O(\log^6 n)$ -approximation algorithm for the demand-robust group Steiner forest problem on weighted graphs with polynomially-bounded aspect ratio.*

Demand-robust group Steiner forest generalizes demand-robust Steiner forest and prior to this work no poly-log approximations were known for demand-robust Steiner forest; thus the above result gives the first poly-log approximation for demand-robust Steiner forest. We solve the tree case of the above problems by observing a connection between demand-robust and online algorithms. In particular, we exploit the fact that for certain online rounding schemes a demand-robust problem can be seen as an online problem with two time steps provided certain natural properties are met. Notably, these properties will be met for these problems *on trees*. Thus, we emphasize that going through the copy tree embedding is crucial for our application – a more direct approach of using online rounding schemes on the general problem does not seem to yield useful results.

Further Applications. Lastly, we note that copy tree embeddings were integral to another recent work of [29], who gave the first poly-log approximations for the hop-constrained version of many classic network design problems, including hop-constrained Steiner forest [1], group Steiner tree and buy-at-bulk network design [4].

2 Graph Notation And Assumptions

Throughout this paper we will work with weighted graphs of the form $G = (V, E, w)$ where V and E are the vertex and edge sets of G and $w : E \rightarrow \mathbb{R}_{\geq 1}$ gives the weight of edges. We typically assume that $n := |V|$ is the number of nodes and write $[n] = \{1, 2, \dots, n\}$. We will also use $V(G)$, $E(G)$ and w_G to stand for the vertex set, edge set and weight function of G . Similarly, we will use w_e to stand for $w(e)$ where convenient. For a subset of edges $F \subseteq E$, we use the notation $w(F) := \sum_{e \in F} w_G(e)$. We use $d_G : V \times V \rightarrow \mathbb{R}_{\geq 0}$ to give the shortest path metric according to w . We will talk about the diameter of a metric (V, d) which is $\max_{u, v \in V} d(u, v)$; we notate the diameter with D . We use $B(v, x) := \{u \in V : d(v, u) \leq x\}$ to stand for the closed ball of v of radius x in metric (V, d) and $B_G(v, x)$ if (V, d) is the shortest path metric of G and we need to disambiguate which graph we are taking balls with respect to. We will sometimes identify a graph with the metric which it induces.

Notice that we have assumed that edge weights are non-zero and at least 1. This will be without loss generality as for our purposes any 0 weight edges may be contracted and scaling of edge weights ensures that the minimum edge weight is at least 1.

3 Online Covering Steiner

In this section we give a deterministic bicriteria algorithm for the online covering Steiner problem which is the same as online group Steiner tree but where we must connect at least r_i vertices from each group g_i to the root. The algorithm is bicriteria in the sense that it relaxes both the r_i -connectivity guarantee and the cost.

As mentioned in the introduction, this problem generalizes group Steiner tree. Moreover, it is also easy to see that any deterministic bicriteria algorithm for online covering Steiner also gives a poly-log-competitive deterministic (unicriteria) algorithm for online (non-group) Steiner tree. In particular, given an instance of Steiner tree on weighted graph $G = (V, E, w)$ with root r where we must connect terminals $A \subseteq V$ to r , it suffices to solve the covering Steiner problem where each vertex in A is in a singleton group with any constant bicriteria relaxation. This is because connecting any $c > 0$ fraction of each group to r will connect at least one vertex to r by the integrality of the number of connected vertices. Thus, our result generalizes the fact that deterministic poly-log approximations are known for online (non-group) Steiner tree [31]. However, we do note that our (deterministic) poly-log-approximate bicriteria online covering Steiner problem algorithm does not imply there is a (deterministic) poly-log-approximate online (non-partial) group Steiner tree algorithm (due to the nature of the bicriteria guarantee).

Offline Covering Steiner Problem. In the covering Steiner problem we are given a weighted graph $G = (V, E, w)$ as well as pairwise disjoint groups $g_1, g_2, \dots, g_k \subseteq V$, desired connected vertices $1 \leq r_i \leq |g_i|$ for each group g_i and root $r \in V$. Our goal is to find a tree T rooted at r which is a subgraph of G and satisfies $|T \cap g_i| \geq r_i$ for every i . We wish to minimize our cost, $w(T) := \sum_{e \in E(T)} w(e)$.³

Online Covering Steiner Problem. The online covering Steiner problem is the same as offline covering Steiner problem but where our solution need not be a tree and groups are revealed in time steps $t = 1, 2, \dots$. That is, in time step t an adversary reveals a new group g_t and the algorithm must maintain a solution T_t where: (1) $T_{t-1} \subseteq T_t$; (2) T_t is feasible for the (offline) covering Steiner tree problem on groups g_1, \dots, g_t and; (3) T_t is cost-competitive with the optimal offline solution for this problem where the cost-competitive ratio of our algorithm is $\max_t w(T_t)/\text{OPT}_t$ where OPT_t is the cost of the optimal offline covering Steiner problem solution on the first t groups. We will give a bicriteria approximation for online covering Steiner; thus we say that an online solution is ρ -connection-competitive if for each t we have $|T_t \cap g_i| \geq r_i \cdot \rho$ for every $i \leq t$.

3.1 Online Covering Steiner on a Tree

We begin by giving a bicriteria deterministic online algorithm for covering Steiner on trees based on a “water-filling” approach. Informally, in iteration t each unconnected vertex in each group will grow the solution towards the root at an equal rate until at least $r_i \cdot (1 - \epsilon)$ vertices in g_t are connected to r .

³ As with group Steiner tree the assumption that the tree is rooted and that the groups are pairwise disjoint is without loss of generality.

3.1.1 Problem

More formally we will solve a problem which is a slight generalization of covering Steiner on trees. We solve this problem on a tree rather than just covering Steiner on a tree because, unlike group Steiner tree, the “groupified” version of covering Steiner is not necessarily another instance of covering Steiner. Roughly, instead of groups we now have groups of groups, hence we call this problem 2-level covering Steiner.

Offline 2-Level Covering Steiner Problem. In the 2-level covering Steiner problem we are given a weighted graph $G = (V, E, w)$, root $r \in V$ and groups of groups $\mathcal{G}_1, \dots, \mathcal{G}_k$ where \mathcal{G}_i consists of groups $\{g_1^{(i)}, \dots, g_{k_i}^{(i)}\}$ where each $g_j^{(i)} \subseteq V$. We are also given connectivity requirements r_1, \dots, r_k . Our goal is to compute a minimum-weight tree T containing r where for each $i \leq k$ we have $|\{g_j^{(i)} : g_j^{(i)} \cap T \neq \emptyset\}| \geq r_i$. We let $n_i := |\{v : \exists j \text{ s.t. } v \in g_j^{(i)}\}|$. Notice that covering Steiner is just 2-level covering Steiner where each $g_i^{(j)}$ is a singleton set.

Online 2-Level Covering Steiner Problem. Online 2-level covering Steiner is the same as the offline problem but where \mathcal{G}_t is revealed in time step t by an adversary. In particular, for each time step t we must maintain a solution T_t where: (1) $T_{t-1} \subseteq T_t$ for all t ; (2) T_t is feasible for the (offline) 2-level covering Steiner problem on $\mathcal{G}_1, \dots, \mathcal{G}_t$ with connectivity requirements r_1, \dots, r_t and; (3) T_t is cost-competitive with the optimal offline solution for this problem where the cost-competitive ratio of our algorithm is $\max_t w(T_t)/\text{OPT}_t$ where OPT_t is the cost of the optimal offline 2-level covering Steiner problem solution on the first t groups of groups.

We will give a bicriteria approximation for online 2-level covering Steiner problem on trees; thus we say that an online solution is ρ -connection-competitive if for each t we have $|\{g_j^{(i)} : g_j^{(i)} \cap T \neq \emptyset\}| \geq \rho \cdot r_i$ for every $i \leq t$.

3.1.2 Algorithm

We now formally describe our algorithm for the 2-level covering Steiner problem on weighted tree $T = (V, E, w)$ given an $\epsilon > 0$. We will maintain a fractional variable $0 \leq x_e \leq w_e$ for each edge indicating the extent to which we buy e where our x_e s will be monotonically increasing as our algorithm runs. Say that an edge e is saturated if $x_e = w_e$.

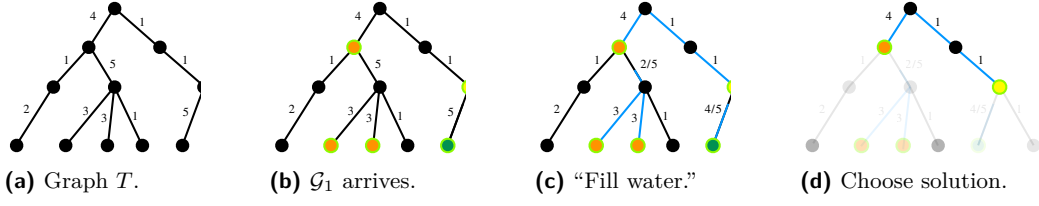
Let us describe how we update our solution in the t th time step. Let T_t be the connected component of all saturated edges containing r . Then, we repeat the following until $|\{g_j^{(t)} : g_j^{(t)} \cap T_t \neq \emptyset\}| \geq r_t \cdot (1 - \epsilon)$. Let $\mathcal{G}'_t := \{g_j^{(t)} \in \mathcal{G}_t : g_j^{(t)} \cap T_t = \emptyset\}$ be all groups in \mathcal{G}_t not yet connected and let $g'_t := \bigcup_{S \in \mathcal{G}'_t} S$ be all vertices in a group which have not yet been connected to r . We say that e is on the frontier for $v \in g'_t$ if it is the first edge on the path from v to r which is not saturated. Similarly, let r_e be the number of vertices in g'_t for which e is on the frontier for v . Then, for each edge e we increase x_e by $r_e \cdot \delta$ where $\delta = \min_e (w_e - x_e)/r_e$. Our solution in the t th time step is T_t once $|\{g_j^{(t)} : g_j^{(t)} \cap T_t \neq \emptyset\}| \geq (1 - \epsilon) \cdot r_t$.

We illustrate one iteration of this algorithm in Figure 3.

3.1.3 Analysis

We proceed to analyze the above algorithm and give its properties.

► **Theorem 11.** *There is a deterministic poly-time algorithm for online 2-level covering Steiner on trees which is $\frac{1}{\epsilon} \cdot (\max_i \frac{n_i}{r_i})$ -cost-competitive and $(1 - \epsilon)$ -connection-competitive.*



■ **Figure 3** Solution our algorithm gives after one group of groups, \mathcal{G}_1 , is revealed where $r_1 = 2$. Nodes in groups in \mathcal{G}_1 outlined in green and nodes colored according to the group of \mathcal{G}_1 which contains them. Saturated edges given in blue and edges with $0 < x_e < w_e$ annotated with “ x_e/w_e ”. All other edges labeled by w_e .

Proof. We begin by verifying that our algorithm returns a monotonically increasing and $(1 - \epsilon)$ -connection-competitive solution. First, notice that our solution is monotonically increasing since our x_e s are monotonically increasing and our solution only includes saturated edges. To see that our solution is $(1 - \epsilon)$ -connection-competitive notice that at least one new edge becomes saturated from each update to the x_e s (namely $\arg \min_e (w_e - x_e)/r_e$) and since if all edges are saturated then $T_t = T$ which clearly satisfies $|\{g_j^{(t)} : g_j^{(t)} \cap T_t \neq \emptyset\}| \geq (1 - \epsilon) \cdot r_t$, this process will eventually halt with a $(1 - \epsilon)$ -connection-competitive solution in the t th iteration. For the same reason our algorithm is deterministic poly-time.

It remains to argue that our solution is $\frac{1}{\epsilon} \cdot (\max_i \frac{n_i}{r_i})$ -cost-competitive. We will argue that we can uniquely charge each unit of increase of our x_e s to an appropriate cost portion of the optimal solution. Fix an iteration t . Next, let $\delta^{(i,j)}$ for $i \leq t$ be the value of δ in the i th iteration the j th time we increase the value of our x_e s. Similarly, let $\delta_x^{(i,j)}$ be the increase in $\sum_e x_e$ when we do so and let $\delta_y^{(i,j)}$ be the increase in $\sum_{e \in T_t^*} x_e$ where T_t^* is the optimal offline solution to the 2-level covering Steiner problem we must solve in the t th iteration. Lastly, let $y := \sum_{i \leq t} \sum_j \delta_y^{(i,j)}$ be the value of $\sum_{e \in T_t^*} x_e$ at the end of the t th iteration; clearly we have $y \leq \text{OPT}_t$. We claim that it suffices to show that for each $i \leq t$ and each j that $\delta_x^{(i,j)} \leq \frac{1}{\epsilon} \delta_y^{(i,j)} \frac{n_i}{r_i}$ since it would follow that at the end of iteration t we have that

$$w(T_t) \leq \sum_e x_e = \sum_{i \leq t} \sum_j \delta_x^{(i,j)} \leq \frac{1}{\epsilon} \sum_{i \leq t} \sum_j \frac{n_i}{r_i} \delta_y^{(i,j)} \leq \frac{1}{\epsilon} \left(\max_i \frac{n_i}{r_i} \right) y \leq \frac{1}{\epsilon} \left(\max_i \frac{n_i}{r_i} \right) \text{OPT}_t.$$

We proceed to show that $\delta_x^{(i,j)} \leq \frac{1}{\epsilon} \delta_y^{(i,j)} \frac{n_i}{r_i}$ for each $i \leq t$ and j . We fix an i and j and for cleanliness of notation we will drop the dependence on i and j in our δ s henceforth.

First, notice that we have that

$$\delta_x \leq n_i \cdot \delta \tag{1}$$

since each vertex $v \in g_i$ is uniquely responsible for up to a δ increase on x_e where e is the edge on v 's frontier.

On the other hand, notice that if a group in \mathcal{G}_i is connected to r by T_t^* but is not yet connected by T_t then such a group uniquely contributes at least δ to δ_y . Since T_t^* connects at least r_i groups in \mathcal{G}_i to r but at the moment of our increase T_t connects at most $(1 - \epsilon) \cdot r_i$, there are at least $\epsilon \cdot r_i$ such groups in \mathcal{G}_i which are connected to r by T_t^* but not by T_t . Thus, we have that

$$\delta_y \geq \epsilon \cdot r_i \cdot \delta \tag{2}$$

Combining Equations 1 and 2 shows $\delta_x \leq \frac{1}{\epsilon} \delta_y \frac{n_i}{r_i}$ as required. ◀

3.2 Online Covering Steiner on General Graphs

Next, we apply our first construction to give an algorithm for covering Steiner on general graphs. Crucially, the following result relies on a single copy tree embedding with polylogarithmic copy number, making our second construction unsuitable for this problem.

► **Theorem 5.** *There is a deterministic poly-time algorithm for online covering Steiner (on general graphs) which is $O\left(\frac{\log^3 n}{\epsilon} \cdot \max_i \frac{|g_i|}{r_i}\right)$ -cost-competitive and $(1 - \epsilon)$ -connection-competitive.*

Proof. We will use our copy tree embedding to produce a single tree on which we must deterministically solve online 2-level covering Steiner. We will then apply the algorithm from Theorem 11 to solve online 2-level covering Steiner on this tree.

More formally, consider an instance of online covering Steiner on weighted graph $G = (V, E, w)$ with root r . Then, we first compute a copy tree embedding $(T, \phi, \pi_{G \rightarrow T}, \pi_{T \rightarrow G})$ deterministically with respect to G and r as in Theorem 3 with cost approximation $O(\log^2 n)$ and copy number $O(\log n)$. Next, given our instance I_t of covering Steiner on G with groups g_1, \dots, g_t and connection requirements r_1, \dots, r_t we let I'_t be the instance of 2-level covering Steiner on T with groups of groups $\mathcal{G}_1, \dots, \mathcal{G}_t$ where $\mathcal{G}_i = \{\phi(v) : v \in g_i\}$, connection requirements r_1, \dots, r_t and root $\phi(r)$. Then if the adversary has required that we solve instance I_t in time step t , then we require that the algorithm in Theorem 11 solves I'_t in time step t and we let H'_t be the solution returned by our algorithm for I'_t . Lastly, we return as our solution for I_t in time step t the set $H_t := \pi_{T \rightarrow G}(H'_t)$.

Let us verify that the resulting algorithm is indeed feasible (i.e. monotone and $(1 - \epsilon)$ -connection-competitive) and of the appropriate cost.

First, we have that $H_t \subseteq H_{t+1}$ for every t since $H'_t \subseteq H'_{t+1}$ because our algorithm for trees returns a feasible solution for its online problem and $\pi_{T \rightarrow G}$ is monotone by definition of a copy tree embedding. Moreover, we claim that H_t connects at least $(1 - \epsilon) \cdot r_i$ vertices from g_i to r for $i \leq t$ and every t . To see this, notice that there are at least $(1 - \epsilon) \cdot r_i$ groups from \mathcal{G}_i containing a vertex connected to r by H'_t . Since each such group consists of the copies of a distinct vertex, by the connectivity preservation properties of a copy tree it follows that H_t connects at least $(1 - \epsilon) \cdot r_i$ vertices from g_i to r .

Next, we verify the cost of our solution. Let OPT'_t be the cost of the optimal solution to I'_t . Notice that since our copy number is $O(\log n)$, it follows that $n_i \leq O(\log n \cdot |g_i|)$. Thus, by the guarantees of Theorem 11 we have

$$w_T(H'_t) \leq \frac{1}{\epsilon} \cdot \left(\max_i \frac{n_i}{r_i} \right) \text{OPT}'_t \leq O\left(\frac{\log n}{\epsilon}\right) \cdot \left(\max_i \frac{|g_i|}{r_i} \right) \text{OPT}'_t. \quad (3)$$

Next, we bound OPT'_t . Let H_t^* be the optimal solution to I_t . We claim that $\pi_{G \rightarrow T}(H_t^*)$ is feasible for I'_t . This follows because H_t^* connects at least r_i vertices from g_i to r for $i \leq t$ and so by the connectivity preservation property of copy tree embeddings we know that there are at least r_i groups in \mathcal{G}_i with a vertex connected to r by $\pi_{G \rightarrow T}(H_t^*)$. Thus, combining this with the $O(\log^2 n)$ cost preservation of our copy tree embedding we have

$$\text{OPT}'_t \leq w_T(\pi_{G \rightarrow T}(H_t^*)) \leq O(\log^2 n) \cdot w_G(H_t^*). \quad (4)$$

Lastly, by the cost preservation property of our copy tree embedding we have that $w_G(H_t) \leq w_T(H'_t)$ which when combined with Equations 3 and 4 gives

$$w_G(H_t) \leq O\left(\frac{\log^3 n}{\epsilon} \cdot \max_i \frac{|g_i|}{r_i}\right) \cdot w_G(H_t^*).$$

thereby showing that our solution is within the required cost bound. ◀

Since group Steiner tree is exactly covering Steiner where $r_i = 1$ in which case $\max_i \frac{|g_i|}{r_i} \leq N$ where again N is the maximum size of a group. Moreover, since any solution can only connect an integral number of vertices from each group, it follows that a $\frac{1}{2}$ -connection-competitive solution for covering Steiner where $r_i = 1$ (i.e. for group Steiner tree) connects at least one vertex from each group. Thus, as a corollary of the above result we have the following deterministic algorithm for online group Steiner tree.⁴

► **Corollary 6.** *There is an $O(N \log^3 n)$ -competitive deterministic algorithm for online group Steiner tree where $N := \max_i |g_i|$ is the maximum group size.*

References

- 1 Ajit Agrawal, Philip Klein, and Ramamoorthi Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.
- 2 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.
- 3 Noga Alon, Richard M Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995.
- 4 Baruch Awerbuch and Yossi Azar. Buy-at-bulk network design. In *Symposium on Foundations of Computer Science (FOCS)*, pages 542–547. IEEE, 1997.
- 5 Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *Symposium on Foundations of Computer Science (FOCS)*, pages 267–276. IEEE, 2011.
- 6 Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Symposium on Foundations of Computer Science (FOCS)*, pages 184–193. IEEE, 1996.
- 7 Yair Bartal. Graph decomposition lemmas and their role in metric embedding methods. In *Annual European Symposium on Algorithms (ESA)*, pages 89–97. Springer, 2004.
- 8 Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog (n)-competitive algorithm for metrical task systems. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 711–719, 1997.
- 9 Yair Bartal and Manor Mendel. Multi-embedding and path approximation of metric spaces. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, volume 3, pages 424–433, 2003.
- 10 Marcin Bienkowski, Bjorn Feldkord, and Pawel Schmidt. A nearly optimal deterministic online algorithm for non-metric facility location. *arXiv preprint*, 2020. [arXiv:2007.07025](https://arxiv.org/abs/2007.07025).
- 11 Niv Buchbinder and Joseph Naor. *The design of competitive online algorithms via a primal-dual approach*. Now Publishers Inc, 2009.
- 12 Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge Plotkin. Approximating a finite metric by a small number of tree metrics. In *Symposium on Foundations of Computer Science (FOCS)*, pages 379–388. IEEE, 1998.
- 13 Kedar Dhamdhere, Vineet Goyal, R Ravi, and Mohit Singh. How to pay, come what may: Approximation algorithms for demand-robust covering problems. In *Symposium on Foundations of Computer Science (FOCS)*, pages 367–376. IEEE, 2005.

⁴ We note that one can use an aforementioned property of our first construction – that if u is connected to r by $F \subseteq E$ then every vertex in $\phi(u)$ is connected to $\phi(r)$ in $\pi_{G \rightarrow T}(F)$ – to reduce the $O(\log^3 n)$ s in this section to $O(\log^2 n)$ s. In particular, if one were to use this property then when we map the solution to our covering Steiner problem on G to our copy tree embedding, the resulting solution will connect at least r_i groups in \mathcal{G}_i at least $\Theta(\log n)$ times. It follows that when we run our water filling algorithm each time it increases $\sum_e x_e$ by 1 we know that it cover at least $\Omega(\log n)$ units of the optimal solution by weight rather than 1 unit of the optimal solution as in the current analysis.

- 14 Matthias Englert and Harald Räcke. Reordering buffers with logarithmic diameter dependency for trees. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1224–1234. SIAM, 2017.
- 15 Matthias Englert, Harald Räcke, and Matthias Westermann. Reordering buffers for general metric spaces. In *Annual ACM Symposium on Theory of Computing (STOC)*, 2007.
- 16 Guy Even, Guy Kortsarz, and Wolfgang Slany. On network design problems: fixed cost flows and the covering steiner problem. *ACM Transactions on Algorithms (TALG)*, 1(1):74–101, 2005.
- 17 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.
- 18 Uriel Feige, Kamal Jain, Mohammad Mahdian, and Vahab Mirrokni. Robust combinatorial optimization with exponential scenarios. In *Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 439–453. Springer, 2007.
- 19 Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM Journal on Computing*, 32(6):1403–1422, 2003.
- 20 Arnold Filtser. Clan embeddings into trees, and low treewidth graphs. *arXiv preprint*, 2021. [arXiv:2101.01146](https://arxiv.org/abs/2101.01146).
- 21 Naveen Garg, Goran Konjevod, and R Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000.
- 22 Daniel Golovin, Vineet Goyal, and R Ravi. Pay today for a rainy day: improved approximation algorithms for demand-robust min-cut and shortest path problems. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 206–217. Springer, 2006.
- 23 Xiangyu Guo, Janardhan Kulkarni, Shi Li, and Jiayi Xian. On the facility location problem in online and dynamic models. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 24 Anupam Gupta, Mohammad T Hajiaghayi, and Harald Räcke. Oblivious network design. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 970–979, 2006.
- 25 Anupam Gupta, Viswanath Nagarajan, and R Ravi. Robust and maxmin optimization under matroid and knapsack uncertainty sets. *ACM Transactions on Algorithms (TALG)*, 12(1):1–21, 2015.
- 26 Anupam Gupta, Viswanath Nagarajan, and Ramamoorthi Ravi. Thresholded covering algorithms for robust and max-min optimization. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 262–274. Springer, 2010.
- 27 Anupam Gupta and Aravind Srinivasan. An improved approximation ratio for the covering steiner problem. *Theory of Computing*, 2(1):53–64, 2006.
- 28 Varun Gupta, Ravishankar Krishnaswamy, and Sai Sandeep. Permutation strikes back: The power of recourse in online metric matching. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2020.
- 29 Bernhard Haeupler, D Ellis Hershkowitz, and Goran Zuzic. Tree embeddings for hop-constrained network design. *Annual ACM Symposium on Theory of Computing (STOC)*, 2021.
- 30 D Ellis Hershkowitz, R Ravi, and Sahil Singla. Prepare for the expected worst: Algorithms for reconfigurable resources under uncertainty. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2019.
- 31 Makoto Imase and Bernard M Waxman. Dynamic steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.
- 32 Richard M Karp. A 2k-competitive algorithm for the circle. *Manuscript*, August, 5, 1989.
- 33 Adam Kasperski and Paweł Zieliński. On the approximability of robust spanning tree problems. *Theoretical Computer Science*, 412(4-5):365–374, 2011.

63:14 Adaptive-Adversary-Robust Algorithms via Small Copy Tree Embeddings

- 34 Rohit Khandekar, Guy Kortsarz, Vahab Mirrokni, and Mohammad R Salavatipour. Two-stage robust network design with exponential scenarios. In *Annual European Symposium on Algorithms (ESA)*, pages 589–600. Springer, 2008.
- 35 Goran Konjevod, Ramamoorthi Ravi, and Aravind Srinivasan. Approximation algorithms for the covering steiner problem. *Random Structures & Algorithms*, 20(3):465–482, 2002.
- 36 Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted steiner tree and related problems. In *Symposium on Foundations of Computer Science (FOCS)*, pages 210–219. IEEE, 2011.
- 37 Harald Racke. Minimizing congestion in general networks. In *Symposium on Foundations of Computer Science (FOCS)*, pages 43–52. IEEE, 2002.

Hedonic Games and Treewidth Revisited

Tesshu Hanaka  

Department of Informatics, Faculty of Information Science and Electrical Engineering,
Kyushu University, Japan

Michael Lampis  

Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France

Abstract

We revisit the complexity of the well-studied notion of Additively Separable Hedonic Games (ASHGs). Such games model a basic clustering or coalition formation scenario in which selfish agents are represented by the vertices of an edge-weighted digraph $G = (V, E)$, and the weight of an arc uv denotes the utility u gains by being in the same coalition as v . We focus on (arguably) the most basic stability question about such a game: given a graph, does a Nash stable solution exist and can we find it efficiently?

We study the (parameterized) complexity of ASHG stability when the underlying graph has treewidth t and maximum degree Δ . The current best FPT algorithm for this case was claimed by Peters [AAAI 2016], with time complexity roughly $2^{O(\Delta^5 t)}$. We present an algorithm with parameter dependence $(\Delta t)^{O(\Delta t)}$, significantly improving upon the parameter dependence on Δ given by Peters, albeit with a slightly worse dependence on t . Our main result is that this slight performance deterioration with respect to t is actually completely justified: we observe that the previously claimed algorithm is incorrect, and that in fact no algorithm can achieve dependence $t^{o(t)}$ for bounded-degree graphs, unless the ETH fails. This, together with corresponding bounds we provide on the dependence on Δ and the joint parameter establishes that our algorithm is essentially optimal for both parameters, under the ETH.

We then revisit the parameterization by treewidth alone and resolve a question also posed by Peters by showing that Nash Stability remains strongly NP-hard on stars under additive preferences. Nevertheless, we also discover an island of mild tractability: we show that Connected Nash Stability is solvable in pseudo-polynomial time for constant t , though with an XP dependence on t which, as we establish, cannot be avoided.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Hedonic Games, Nash Equilibrium, Treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.64

Related Version *Full Version:* <https://arxiv.org/abs/2202.06925>

Funding This work is partially supported by PRC CNRS JSPS project PARAGA (Parameterized Approximation Graph Algorithms) JPJSBP 120192912 and by JSPS KAKENHI Grant Number JP21K17707, JP21H05852, JP22H00513.

Acknowledgements We want to thank Dr. Rémy Belmonte for helpful discussions.

1 Introduction

Coalition formation is a topic of central importance in computational social choice and in the mathematical social sciences in general. The goal of its study is to understand how groups of selfish agents are likely to partition themselves into teams or clusters, depending on their preferences. The most well-studied case of coalition formation are *hedonic games*, which have the distinguishing characteristic that each agent's utility only depends on the coalition on which she is placed (and not on the coalitions of other players). Hedonic games



© Tesshu Hanaka and Michael Lampis;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 64; pp. 64:1–64:16
Leibniz International Proceedings in Informatics



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

■ **Table 1** Summary of results. t, p, Δ, W denote the treewidth, pathwidth, maximum degree, and maximum absolute weight. Results denoted by (G) apply to general (possibly disconnected) NASH STABILITY, and by (C) to CONNECTED NASH STABILITY.

Parameter	Algorithms	Lower Bounds
t, p	$(nW)^{O(t^2)}$ (C) (Theorem 11)	Strongly NP-hard for Stars (G) (Theorem 9) No $f(p) \cdot n^{o(p/\log p)}$ (C) (Theorem 12)
$t, p + \Delta$	$(\Delta t)^{O(\Delta t)} (n + \log W)^{O(1)}$ (G) (Theorem 4)	No $(p\Delta)^{o(p\Delta)} (nW)^{O(1)}$ (G) (Theorem 5) No $\Delta^{o(\Delta)} (nW)^{O(1)}$ if $p = O(1)$ (G) (Corollary 6) No $p^{o(p)} n^{O(1)}$ if $\Delta, W = O(1)$ (Theorem 7) (G,C)

have recently been an object of intense study also from the computer science perspective [1, 2, 6, 7, 9, 10, 12, 19, 27, 33, 40], due in part to their numerous applications in, among others, social network analysis [34], scheduling group activities [15], and allocating tasks to wireless agents [39]. For more information we refer the reader to [13] and the relevant chapters of standard computational social choice textbooks [4].

Hedonic games are extremely general and capture many interesting scenarios in algorithmic game theory and computational social choice. Unfortunately, this generality implies that most interesting questions about such games are computationally hard; indeed, even encoding the preferences of agents generally takes exponential space. This has motivated the study of natural succinctly representable versions of hedonic games. In this paper, we focus on one of the most widely-studied such models called Additively-Separable Hedonic Games (ASHG). In this setting the interactions between agents are given by an edge-weighted directed graph $G = (V, E)$, where the weight of an arc $uv \in E$ denotes the utility that u gains by being placed in the same coalition as v . Thus, vertices which are not connected by an arc are considered to be indifferent to each other. Given a partition into coalitions, the utility of a player v is defined as the sum of the weights of out-going arcs from v to its own coalition.

A rich literature exists studying various questions about ASHGs, including a large spectrum of stability concepts and social welfare maximization [3, 5, 17, 20, 23, 34, 35, 41]. In this paper we focus on perhaps the most basic notion of stability one may consider. We say that a configuration π is *Nash Stable* if no agent v can unilaterally strictly increase her utility by selecting a different coalition of π or by forming a singleton coalition. The algorithmic question that we are interested in studying is the following: given an ASHG, does a Nash Stable partition exist? Even though other notions of stability exist (notably when deviating players are allowed to collaborate [11, 16, 37, 42]), fully understanding the complexity of NASH STABILITY is of particular importance, because of the fundamental nature of this notion.

NASH STABILITY of ASHGs has been thoroughly studied and is, unfortunately, NP-complete. We therefore adopt a parameterized point of view and investigate whether some desirable structure of the input can render the problem tractable. We consider two of the most well-studied graph parameters: the treewidth t and the maximum degree Δ of the underlying graph. The study of ASHGs in this light was previously taken up by Peters [36] and the goal of our paper is to improve and clarify the state of the art given by this previous work.

Summary of Results. Our results can be divided into two parts (see Table 1 for a summary). In the first part of the paper we parameterize the problem by $t + \Delta$, that is, we study its complexity for graphs that have simultaneously low treewidth and low maximum degree.

The study of hedonic games on such graphs was initiated by Peters [36], who already considered a wide variety of algorithmic questions on ASHG for these parameters and provided FPT algorithms using Courcelle’s theorem. Due to the importance of NASH STABILITY, more refined algorithmic arguments were given in the same work, and it was claimed that CONNECTED NASH STABILITY (the variant of the problem where coalitions must be connected in the underlying graph) and NASH STABILITY can be decided with parameter dependence roughly $2^{\Delta^2 t}$ and $2^{\Delta^5 t}$, respectively (though as we explain below, these claims were not completely justified). We thus revisit the problem with the goal of determining the optimal parameter dependence for NASH STABILITY in terms of t and Δ . Our positive contribution is an algorithm deciding NASH STABILITY in time $(\Delta t)^{O(\Delta t)} (n + \log W)^{O(1)}$, where W is the maximum absolute weight, significantly improving the parameter dependence for Δ (Theorem 4). This is achieved by reformulating the problem as a coloring problem with $t\Delta$ colors in a way that encodes the property that two vertices belong in the same coalition and then using dynamic programming to solve this problem. Our main technical contribution is then to establish that our algorithm is essentially optimal. To that end we first show that if there exists an algorithm solving NASH STABILITY in time $(p\Delta)^{o(p\Delta)} (nW)^{O(1)}$, where p is the pathwidth of the underlying graph, then the ETH is false (Theorem 5). Hence, it is not possible to obtain a better parameter dependence, even if we accept a pseudo-polynomial running time and a more restricted parameter.

If we were considering a parameterization with a single parameter, at this point we would be essentially done, since we have an algorithm and a lower bound that match. However, the fact that Δ and t are two a priori independent variables significantly complicates the analysis because, informally, the space of running time functions that depend on two variables is not totally ordered. To see what we mean by that, recall that [36] claimed an algorithm with complexity roughly $2^{\Delta^5 t}$, while our algorithm’s complexity has the form $(\Delta t)^{\Delta t}$. The two algorithms are not directly comparable in performance: for some values of Δ, t one is better and for some the other (though the range of parameters where $2^{\Delta^5 t} < (\Delta t)^{\Delta t}$ is quite limited). As a result, even though Theorem 5 shows that no algorithm can beat the algorithm of Theorem 4 in all cases, it does not rule out the possibility that some algorithm beats it in *some* cases, for example when Δ is much smaller than t , or vice-versa. We therefore need to work harder to argue that our algorithm is indeed optimal in essentially all cases. In particular, we show that even if pathwidth is constant the problem cannot be solved in $\Delta^{o(\Delta)} (nW)^{O(1)}$ (Corollary 6); and even if Δ and W are constant, the problem cannot be solved in $p^{o(p)} n^{O(1)}$ (Theorem 7) unless the ETH is false. Hence, we succeed in covering essentially all corner cases, showing that our algorithm’s slightly super-exponential dependence on *the product* of Δ and t is truly optimal, and we cannot avoid the slightly super-exponential on either parameter, even if we were to accept a much worse dependence on the other.

An astute reader will have noticed a contradiction between our lower bounds and the algorithms of [36]. It is also worth noting that Theorem 7 applies to both the connected and disconnected cases of the problem, using an argument due to [36]. Hence, Theorem 7 implies that, either the ETH is false, or *neither* of the aforementioned algorithms of [36] can have the claimed performance, as executing them on the instances produced by our reduction (which have $\Delta = O(1)$) would give parameter dependence $2^{O(t)}$, which is ruled out by Theorem 7. Indeed, in Section 3 we explain in more detail that the argumentation of [36] lacks an ingredient (the partition of vertices in each neighborhood into coalitions) which turns out to be necessary to obtain a correct algorithm and also key in showing the lower bound. Hence, the slightly super-exponential dependence on t cannot be avoided (under the ETH), and the dependence on t promised in [36] is impossible to achieve: the best one can hope for is the slightly super-exponential dependence on both t and Δ given in Theorem 4.

In the second part of the paper, we consider NASH STABILITY on graphs of low treewidth, without making any further assumptions (in particular, we consider graphs of arbitrarily large degree). This parameterization was considered by Peters [36] who showed that the problem is strongly NP-hard on stars and thus motivated the use of the double parameter $t + \Delta$. This would initially appear to settle the problem. However, we revisit this question and make two key observations: first, the reduction of [36] does not show hardness for additive games, but for a more general version of the problem where preferences of players are not necessarily additive but are described by a collection of boolean formulas (HC-nets [18, 25]). It was therefore explicitly posed as an open question whether *additive* games are also hard [36]. Second, in the reduction of [36] coalitions are disconnected. As noted in [26, 36], there are situations where Nash Stable coalitions make more sense if they are connected in the underlying graph. We therefore ask whether CONNECTED NASH STABILITY, where we impose a connectivity condition on coalitions, is an easier problem.

Our first contribution is to resolve the open question of [36] by showing that imposing either one of these two modifications does *not* render the problem tractable: NASH STABILITY of additive hedonic games is still strongly NP-hard on stars (Theorem 9); and CONNECTED NASH STABILITY of hedonic games encoded by HC-nets is still NP-hard on stars (Theorem 10). However, our reductions stubbornly refuse to work for the natural combination of these conditions, namely, CONNECTED NASH STABILITY for additive hedonic games on stars. Surprisingly, we discover that this is with good reason: CONNECTED NASH STABILITY turns out to be solvable in pseudopolynomial time on graphs of bounded treewidth (Theorem 11). More precisely, our algorithm, which uses standard dynamic programming techniques but crucially relies on the connectedness of coalitions, runs in “pseudo-XP” time, that is, in polynomial time when $t = O(1)$ and weights are polynomially bounded. Completing our investigation we show that this is essentially the best possible: obtaining a pseudo-polynomial time algorithm with FPT dependence on treewidth (or pathwidth) would contradict standard assumptions (Theorem 12). Hence, in this part we establish that there is an overlooked case of ASHG that does become somewhat tractable when we only parameterize by treewidth, but this tractability is limited.

Related work. Deciding if an ASHG admits a partition that is Nash Stable or has other desirable properties is NP-hard [3, 5, 34, 38, 41]. Hardness remains even in cases where a Nash Stable solution is guaranteed, such as symmetric preferences, where the problem is PLS-complete [21], and non-negative preferences, where it is NP-hard to find a non-trivial stable partition [35]. The problem generally remains hard when we impose the requirement that coalitions must be connected [8, 26].

A related MIN STABLE CUT problem is studied in [29], where we partition the vertices into two coalitions in a Nash Stable way. Interestingly, the complexity of that problem turns out to be $2^{O(\Delta t)}$, since each vertex has 2 choices; this nicely contrasts with NASH STABILITY, where vertices have more choices, and which is slightly super-exponential parameterized by treewidth. Similar slightly super-exponential complexities have been observed with other problems involving treewidth and partitioning vertices into sets [24, 32].

2 Preliminaries

We use standard graph-theoretic notation and assume that the reader is familiar with standard notions in parameterized complexity, including treewidth t and pathwidth p [14]. We mostly deal with directed graphs and denote an arc from vertex u to vertex v as uv .

When we talk about the degree or the neighborhood of a vertex v , we refer to its degree and its neighborhood in the underlying graph, that is, the graph obtained by forgetting the directions of all arcs. Throughout the paper $\Delta(G)$ (or simply Δ , when G is clear from the context) denotes the maximum degree of the underlying graph of G . The Exponential Time Hypothesis (ETH) is the assumption that there exists $c > 1$ such that 3-SAT on formulas with n variables does not admit a c^n algorithm [28]. We will mostly use a somewhat simpler to state (and weaker) form of this assumption stating that 3-SAT cannot be solved in time $2^{o(n)}$.

In this paper we will be mostly interested in *Additively Separable Hedonic Games* (ASHG). In an ASHG we are given a directed graph $G = (V, E)$ and a weight function $w : V \times V \rightarrow \mathbb{Z}$ that encodes agents' preferences. The function w has the property that for all $u, v \in V$ such that $uv \notin E$ we have $w(u, v) = 0$, that is, non-zero weights are only given to arcs. A solution to an ASHG is a partition π of V , where we refer to the sets of V as classes or, more simply, as coalitions. For each $v \in V$ and $S \subseteq V$ the utility that v derives from being placed in the coalition S is defined as $p_v(S) = \sum_{u \in S \setminus \{v\}} w(v, u)$. A partition π is Nash Stable if we have the following: for each $v \in V$, if v belongs in the class S of π , we have $p_v(S) \geq 0$ and for each $S' \in \pi$ we have $p_v(S) \geq p_v(S')$. In other words, no vertex can strictly increase its utility by joining another coalition of π or forming a singleton coalition. We also consider the notion of *Connected Nash Stable* partitions, which are Nash Stable partitions π with the added property that all classes of π are connected in the underlying undirected graph of G .

3 Parameterization by Treewidth and Degree

In this section we revisit NASH STABILITY parameterized by $t + \Delta$, which was previously studied in [36]. Our main positive result is an algorithm given in Section 3.1 solving the problem with dependence $(t\Delta)^{O(t\Delta)}$.

Our main technical contribution is then to show in Section 3.2 that this algorithm is essentially optimal, under the ETH. As explained, we need several different reductions to settle this problem in a satisfactory way. The main reduction is given in Theorem 5 and uses the fact that a partition restricted to the neighborhood of a vertex with degree Δ encodes roughly $\Delta \log \Delta$ bits of information, because there are around Δ^Δ partitions of Δ elements into equivalence classes. This key idea allows the first reduction to compress the treewidth more and more as Δ increases. Hence, we can produce instances where both t and Δ are super-constant, but appropriately chosen to match our bound. In this way, Theorem 5 rules out running times of the form, say $(t\Delta)^{t+\Delta}$, as when t, Δ are both super-constant, $t + \Delta = o(t\Delta)$. By modifying the parameters of Theorem 5 we then obtain Corollary 6 from the same construction, which states that no algorithm can have dependence $\Delta^{o(\Delta)}$, even on graphs of bounded pathwidth. On the other hand, this type of construction cannot show hardness for instances of bounded degree, as when $\Delta = O(1)$, then $\Delta^\Delta = O(1)$, so we cannot really compress the treewidth of the produced instance. Hence, we use a different reduction in Theorem 7, showing that the problem cannot be solved with dependence $p^{o(p)}$ on instances of bounded degree. This reduction uses a super-constant number of coalitions that “run through” the graph, and hence produces instances with super-constant t . The three complementary reductions together cover the whole range of possibilities and indicate that there is not much room for improvement in our algorithm.

It is worth discussing here that, assuming the ETH, Theorem 7 contradicts the claimed algorithms of [36], which for $\Delta = O(1)$ would solve (CONNECTED) NASH STABILITY with dependence $2^{O(t)}$, while Theorem 7 claims that the problem cannot be solved in time $2^{o(t \log t)}$.

Let us then briefly explain why the proof sketch for these algorithms in [36] is incomplete: the idea of the algorithms is to solve CONNECTED NASH STABILITY, and use the arcs of the instance to verify connectivity. Hence, the DP algorithm will remember, in a ball of distance 2 around each vertex, which arcs have both of their endpoints in the same coalition. The claim is that this information allows us to infer the coalitions. Though this is true if one is given this information for the whole graph, it is not true locally around a vertex where we only have information about other vertices which are close by. In particular, it could be the case that u has neighbors v_1, v_2 , which happen to be in the same coalition, but such that the path proving that this coalition is connected goes through vertices far from u . Because this cannot be verified locally, any DP algorithm would need to store some connectivity information about the vertices in a bag which, as implied by Theorem 7 inevitably leads to a dependence of the form t^t .

3.1 Improved FPT Algorithm

In order to obtain our algorithm for NASH STABILITY we will need two ingredients. The first ingredient will be a reformulation of the problem as a vertex coloring problem. We use the following definition where, informally, a vertex is stable if its outgoing weight to vertices of the same color cannot be increased by changing its color.

► **Definition 1.** *A Stable k -Coloring of an edge-weighted digraph G is a function $c : V \rightarrow [k]$ satisfying the following property: for each $v \in V$ we have $\sum_{u \in c^{-1}(c(v))} w(v, u) \geq \max_{j \in [k+1]} \sum_{u \in c^{-1}(j)} w(v, u)$.*

Note that in the definition above we take the maximum over $j \in [k+1]$ of the total weight of v towards color class j . Since c is a function that uses k colors, we have $c^{-1}(k+1) = \emptyset$ and hence this ensures that the total weight of v towards its own color must always be non-negative in a stable coloring. Also note that to calculate the total weight from v to a certain color class j , it suffices to consider the vertices of color j that belong in the out-neighborhood of v .

Our strategy will be to show that, for appropriately chosen k , deciding whether a graph admits a stable k -Coloring is equivalent to deciding whether a Nash Stable partition exists. Then, the second ingredient of our approach is to use standard dynamic programming techniques to solve Stable k -Coloring on graphs of bounded treewidth and maximum degree.

The key lemma for the first part is the following:

► **Lemma 2.** *Let $G = (V, E)$ be an edge-weighted digraph whose underlying graph has maximum degree Δ and admits a tree decomposition with maximum bag size t . Then, G has a Nash Stable partition if and only if it admits a Stable k -Coloring for $k = t \cdot \Delta$.*

Proof. First, suppose that we have a Stable k -Coloring $c : V \rightarrow [k]$ of the graph for some value k . We obtain a Nash Stable partition of $V(G)$ by turning each color class into a coalition. By the definition of Stable k -Coloring, each vertex has at least as high utility in its own color class (and hence its own coalition) as in any other, so this partition is stable.

For the converse direction, suppose that there exists a Nash Stable partition π of G . We will first attempt to color the coalitions of π in a way that any two coalitions which are at distance at most two receive distinct colors, while using at most $t \cdot \Delta$ colors. In the remainder, when we refer to the distance between two sets of vertices S_1, S_2 , we mean $\min_{u \in S_1, v \in S_2} d(u, v)$, where distances are calculated in the underlying graph.

Consider the graph G^2 obtained from the underlying graph of G by connecting any two vertices which are at distance at most 2 in the underlying graph of G . We can construct a tree decomposition of G^2 where all bags contain at most $t \cdot \Delta$ vertices by taking the assumed tree decomposition of G and adding to each bag the neighbors of all vertices contained in

that bag. Furthermore, we can assume without loss of generality that any equivalence class C of the Nash Stable partition π is connected in G^2 . If not, that would mean that there exists a class C that contains a connected component $C' \subseteq C$ such that C' is at distance at least 3 from $C \setminus C'$ in the underlying graph of G . In that case we could partition C into two classes $C', C \setminus C'$, without affecting the stability of the partition.

Formally now the claim we wish to make is the following:

▷ **Claim 3.** There is a coloring c of the equivalence classes of π with $k = t \cdot \Delta$ colors such that any two classes C_1, C_2 of π which are at distance at most two in the underlying graph of G receive distinct colors.

Proof. We prove the claim by induction on the number of equivalence classes of π . If there is only one class the claim is trivial.

Consider a rooted tree decomposition of G^2 . For an equivalence class C of π we say that the bag B is the top bag for C if B contains a vertex of C and no bag that is closer to the root contains a vertex of C . Select an equivalence class C of π whose top bag is as far from the root as possible. We claim that there are at most $t \cdot \Delta - 1$ classes C' which are at distance at most 2 from C in G .

In order to prove that there are at most $t \cdot \Delta - 1$ other classes at distance at most two from C , consider such a class C' , which is therefore at distance one from C in G^2 . Let B be the top bag of C . If C' does not contain any vertex that appears in B then we get a contradiction as follows: first, C' has a neighbor of a vertex of C , so these two vertices must appear together in a bag; since all vertices of C appear in the sub-tree rooted at B , some vertices of C' must appear strictly below B in the decomposition; since B is a separator of G^2 and C' is connected, if no vertex of C' is in B then all vertices of C' appear below B in the decomposition; but then, this contradicts the choice of C as the class whose top bag is as far from the root as possible. As a result, for each C' that is a neighbor of C in G^2 , there exists a distinct vertex of C' in B . Since $|B| \leq t \cdot \Delta$ and B contains a vertex of C , we get that the coalitions C' which are neighbors of C in G^2 are at most $t \cdot \Delta - 1$.

We now remove all vertices of C from the graph and claim that π restricted to the new graph is still a Nash Stable partition. By induction, there is a coloring of the remaining coalitions of π that satisfies the claim. We keep this coloring and assign to C a color that is not used by any of the at most $k - 1$ coalitions which are at distance two from C . Hence, we obtain the claimed coloring of the classes of π . ◀

From Claim 3 we obtain a coloring of the equivalence classes of π with $k = t \cdot \Delta$ colors, such that any two equivalence classes which are at distance at most 2 in the underlying graph of G receive distinct colors. We now obtain a coloring of V by assigning to each vertex the color of its class. In the out-neighborhood of each vertex v the partition induced by the coloring is the same as that induced by π , since all the vertices in the out-neighborhood of v are at distance at most 2 from each other in G . Hence, the k -Coloring must be stable, because otherwise a vertex would have incentive to deviate in π by joining another coalition or by becoming a singleton. ◀

► **Theorem 4.** *There exists an algorithm which, given an ASHG defined on a digraph $G = (V, E)$ whose underlying graph has maximum degree Δ and a tree decomposition of the underlying graph of G of width t , decides if a Nash Stable partition exists in time $(\Delta t)^{O(\Delta t)} (n + \log W)^{O(1)}$, where $n = |V|$ and W is the largest absolute weight.*

Proof. Using Lemma 2 we will formulate an algorithm that decides if the given instance admits a Stable k -Coloring for $k = (t + 1)\Delta$, since this is equivalent to deciding if a Nash Stable partition exists. We first obtain a tree decomposition of G^2 by placing into each bag of the given decomposition all the neighbors of all the vertices of the bag.

We now execute a standard dynamic programming algorithm for k -coloring on this new decomposition, so we sketch the details. The DP table has size $k^{(t+1)\Delta} = (\Delta t)^{O(\Delta t)}$ since we need to store as a signature of a partial solution the colors of all vertices contained in a bag. The only difference with the standard DP algorithm for coloring is that our algorithm, whenever a new vertex v is introduced in a bag B , considers all possible colors for v , and then for each $u \in B$, if all neighbors of u are contained in B , verifies for each signature whether u is stable. Signatures where a vertex is not stable are discarded. The key property is now that for any vertex u , there exists a bag B such that B contains u and all its neighbors (since in G^2 the neighborhood of u is a clique), hence only signatures for which all vertices are stable may survive until the root of the decomposition. ◀

3.2 Tight ETH-based Lower Bounds

► **Theorem 5.** *If the ETH is true, there is no algorithm which decides if an ASHG on a graph with n vertices, maximum degree Δ , and pathwidth p admits a Nash Stable partition in time $(p\Delta)^{o(p\Delta)}(nW)^{O(1)}$, where W is the maximum absolute weight.*

Proof. We will give a parametric reduction which, starting from a 3-SAT instance ϕ with n variables and m clauses, and for any desired parameter $d < n/\log n$, constructs an ASHG instance G with the following properties:

1. G can be constructed in time polynomial in n
2. G has maximum degree $O(d)$
3. G has pathwidth $O(\frac{n}{d \log d})$
4. the maximum absolute value W is $2^{O(d)}$
5. ϕ is satisfiable if and only if there exists a Nash Stable partition.

Before we go on, let us argue why a reduction that satisfies these properties does indeed establish the theorem: given a 3-SAT instance on n variables, we set $d = \lfloor \sqrt{n} \rfloor$. We construct G in polynomial time, therefore the size of G is polynomially bounded by n . Deciding if G has a Nash Stable partition is equivalent to solving ϕ by the last property. By the third property, the pathwidth of the constructed graph is $O(\frac{\sqrt{n}}{\log n})$, so $p\Delta = O(\frac{n}{\log n})$. Furthermore, $W = 2^{O(\sqrt{n})}$. If deciding if a Nash Stable partition exists can be done in time $(p\Delta)^{o(p\Delta)}(|G| \cdot W)^{O(1)}$, the total running time for deciding ϕ is $(p\Delta)^{o(p\Delta)}(|G| \cdot W)^{O(1)} = 2^{o(n)}$ contradicting the ETH.

We now describe our construction. We are given a 3-SAT instance ϕ with variables x_0, \dots, x_{n-1} , and a parameter d , which we assume to be a power of 2 (otherwise we increase its value by at most a factor of 2). We also assume without loss of generality that all clauses of ϕ have size exactly 3 (otherwise we repeat literals). We construct the following graph:

1. **Selection vertices:** for each $i_1 \in \{0, \dots, \lceil \frac{n}{d \log d} \rceil\}$, $i_2 \in \{0, \dots, d-1\}$, $j \in \{0, \dots, m\}$, we construct a vertex $u_{(i_1, i_2, j)}$.
2. **Consistency vertices:** for each $i_1 \in \{0, \dots, \lceil \frac{n}{d \log d} \rceil\}$, $j \in \{1, \dots, m-1\}$, we construct a vertex $c_{(i_1, j)}$. For $i_2 \in \{0, \dots, d-1\}$ we give weights: $w(c_{(i_1, j)}, u_{(i_1, i_2, j)}) = 4^{i_2}$; $w(c_{(i_1, j)}, u_{(i_1, i_2, j+1)}) = -4^{i_2}$; $w(u_{(i_1, i_2, j)}, c_{(i_1, j)}) = w(u_{(i_1, i_2, j+1)}, c_{(i_1, j)}) = -4^d$.
3. **Clause gadget:** for each $j \in \{1, \dots, m\}$ we construct two vertices s_j, s'_j and set $w(s_j, s'_j) = 2$. We also construct three vertices $\ell_{(j,1)}, \ell_{(j,2)}, \ell_{(j,3)}$ and set $w(\ell_{(j,1)}, s_j) = w(\ell_{(j,2)}, s_j) = w(\ell_{(j,3)}, s_j) = 2$ and $w(s_j, \ell_{(j,1)}) = w(s_j, \ell_{(j,2)}) = w(s_j, \ell_{(j,3)}) = -1$.
4. **Palette gadget:** we construct a vertex p and a helper p' . We set $w(p, p') = w(p', p) = 1$. Furthermore, for $i_1 = \lceil \frac{n}{d \log d} \rceil$ and for all $i_2 \in \{0, \dots, d-1\}$, we set $w(p, u_{(i_1, i_2, 0)}) = 1$ and $w(u_{(i_1, i_2, 0)}, p) = -1$. We call selection vertex $u_{(i_1, i_2, 0)}$ a *palette vertex*.

So far, we have described the main part of our construction, without yet specifying how we encode which literals appear in each clause. Before we move on to describe this part, let us give some intuition about the construction up to this point. The intended meaning of the palette gadget is that vertices $u_{(i_1, i_2, 0)}$ for $i_1 = \lceil \frac{n}{d \log d} \rceil$ and $i_2 \in \{0, \dots, d-1\}$ should be placed in distinct coalitions (p can be thought of as a stalker). These vertices form a “palette”, in the sense that every other selection vertex encodes an assignment to some of the variables of ϕ by deciding which of the palette vertices it will join. Hence, we intend to extract an assignment of ϕ from a stable partition by considering each vertex $u_{(i_1, i_2, 0)}$, for $i_1 \in \{0, \dots, \lceil \frac{n}{d \log d} \rceil - 1\}$, $i_2 \in \{0, \dots, d-1\}$. For each such vertex we test in which of the d palette partitions the vertex was placed, and this gives us enough information to encode $\log d$ variables of ϕ . Since we have $\lceil \frac{n}{d \log d} \rceil \cdot d \geq \frac{n}{\log d}$ non-palette selection vertices, and each such selection vertex encodes $\log d$ variables, we will be able to encode an assignment to n variables. The role of the consistency vertices is to make sure that the partition of the selection vertices (and hence, the encoded assignment) stays consistent throughout our construction.

In order to complete the construction, let us make the above intuition more formal. For $i_1 \in \{0, \dots, \lceil \frac{n}{d \log d} \rceil - 1\}$, $i_2 \in \{0, \dots, d-1\}$ and for any $j \in \{1, \dots, m\}$, we will say that $u_{(i_1, i_2, j)}$ encodes the assignment to variables x_k , with $k \in \{i_1 \cdot d \log d + i_2 \log d, \dots, i_1 \cdot d \log d + i_2 \log d + \log d - 1\}$. Equivalently, given an integer k , we can compute which selection vertices encode the assignment to x_k by setting $i_1 = \lfloor \frac{k}{d \log d} \rfloor$ and $i_2 = \lfloor \frac{k - i_1 d \log d}{\log d} \rfloor$. In that case, x_k is represented by $u_{(i_1, i_2, j)}$ (for any j).

Let us now explain precisely how an assignment to the variables of ϕ is encoded by the placement of selection vertices in coalitions. Let k be such that x_k is encoded by $u_{(i_1, i_2, j)}$ and let $i_3 = k - i_1 d \log d - i_2 \log d$. We have $i_3 \in \{0, \dots, \log d - 1\}$. If x_k is set to True in the assignment, then $u_{(i_1, i_2, j)}$ must be placed in the same coalition as a palette vertex $u_{\lceil \frac{n}{d \log d} \rceil, i'_2, 0}$ where i'_2 has the following property: if we write i'_2 in binary, then the bit in position i_3 must be set to 1. Similarly, if x_k is set to False, then we must place $u_{(i_1, i_2, j)}$ in the same coalition as a palette vertex $u_{\lceil \frac{n}{d \log d} \rceil, i'_2, 0}$ where writing i'_2 in binary gives a 0 in position i_3 . Observe that, given an assignment and a vertex $u_{(i_1, i_2, j)}$ which represents $\log d$ variables, this process fully specifies the palette vertex with which we must place $u_{(i_1, i_2, j)}$ to represent the assignment. In the converse direction, we can extract from the placement of $u_{(i_1, i_2, j)}$ an assignment to the vertices it represents if we know that all palette vertices are placed in distinct components, simply by finding the palette vertex $u_{(\lceil \frac{n}{d \log d} \rceil, i'_2, 0)}$ in the coalition of $u_{(i_1, i_2, j)}$, writing down i'_2 in binary, and using its $\log d$ bits in order to give an assignment to the $\log d$ variables represented by $u_{(i_1, i_2, j)}$.

We are now ready to complete the construction by considering each clause. Each vertex $\ell_{(j, \alpha)}$, $\alpha \in \{1, 2, 3\}$, corresponds to a literal of the j -th clause of ϕ . If this literal involves the variable x_k , we calculate integers i_1, i_2, i_3 from k as explained in the previous paragraph. Say, x_k is the i_3 -th variable represented by $u_{(i_1, i_2, j)}$. We set $w(\ell_{(j, \alpha)}, u_{(i_1, i_2, j)}) = 1$. Furthermore, for each $i'_2 \in \{0, \dots, d-1\}$ we look at the i_3 -th bit of the binary representation of i'_2 . If setting x_k to the value of that bit would make the literal represented by $\ell_{(j, \alpha)}$ True, we set $w(\ell_{(j, \alpha)}, u_{(\lceil \frac{n}{d \log d} \rceil, i'_2, j)}) = 1$; otherwise we set $w(\ell_{(j, \alpha)}, u_{(\lceil \frac{n}{d \log d} \rceil, i'_2, j)}) = 0$. We perform the above process for all $j \in \{1, \dots, m\}$, $\alpha \in \{1, 2, 3\}$.

Our construction is now complete, so we need to show that we satisfy all the claimed properties. It is not hard to see that the graph can be built in polynomial time, and the maximum absolute weight used is $2^{O(d)}$ (on arcs incident on some consistency vertices). The vertices with maximum degree are the consistency vertices and the vertices representing literals, both of which have degree $O(d)$.

To establish the bound on the pathwidth we first delete p, p' from the graph, as this can decrease pathwidth by at most 2. Now observe that, for each j , the set $C_j = \{c_{(i_1, j)} \mid i_1 \in \{0, \dots, \lceil \frac{n}{d \log d} \rceil\}\}$ is a separator of the graph. We claim that if we fix a j , then the set $C_j \cup C_{j+1}$ separates the set $C'_j = \{u_{(i_1, i_2, j)} \mid i_1 \in \{0, \dots, \lceil \frac{n}{d \log d} \rceil\}, i_2 \in \{0, \dots, d-1\}\} \cup \{s_j, s'_j, \ell_{(j,1)}, \ell_{(j,2)}, \ell_{(j,3)}\}$ from the rest of the graph. We claim that we can calculate a path decomposition of the graph induced by $C_j \cup C'_j \cup C_{j+1}$ with width $O(\frac{n}{d \log d})$ such that the first bag contains C_j and the last bag contains C_{j+1} . If we achieve this we can construct a path decomposition of the whole graph by gluing these decompositions together in the obvious way (in order of increasing j). However, a path decomposition of this induced subgraph can be constructed by placing $C_j \cup C_{j+1} \cup \{s_j, s'_j, \ell_{(j,1)}, \ell_{(j,2)}, \ell_{(j,3)}\}$ and a distinct vertex of the remainder of C'_j in each bag. This decomposition has width $2|C_j| + O(1) = O(\frac{n}{d \log d})$.

Finally, let us establish the main property of the construction, namely that ϕ is satisfiable if and only if the ASHG instance admits a Nash Stable partition. If there exists a satisfying assignment to ϕ we construct a partition as follows: (i) p, p' are in their own coalition (ii) each consistency vertex is a singleton (iii) for $i_2 \in \{0, \dots, d-1\}$, the vertices of $\{u_{\lceil \frac{n}{d \log d} \rceil, i_2, j} \mid j \in \{1, \dots, m\}\}$ are placed in a distinct coalition (iv) we place the remaining selection vertices in one of the previous d coalitions in a way that represents the assignment as previously explained (v) for each $j \in \{1, \dots, m\}$ the j -th clause contains a True literal; we place the corresponding vertex $\ell_{(j, \alpha)}$ together with its out-neighbor in the selection vertices, and the remaining literal vertices together with s, s' in a new coalition. We claim that this partition is Nash Stable. We have the following argument: (i) p' is with p , while p cannot increase her utility by leaving p' , since all its other out-neighbors are in distinct coalitions (ii) for each i_1, i_2, j , the vertices $u_{(i_1, i_2, j)}, u_{(i_1, i_2, j+1)}$ are in the same coalition. Hence, the utility of each consistency vertex is 0 in any coalition, and such vertices are stable as singletons (iii) each selection vertex $u_{(i_1, i_2, j)}$ has utility 0, and such vertices only have out-going arcs of negative weight (iv) in each clause gadget we have a coalition with s_j, s'_j together with two literal vertices, say $\ell_{(j,1)}, \ell_{(j,2)}$; no vertex has incentive to leave this coalition (v) finally, for literal vertices $\ell_{(j, \alpha)}$ which we placed together with a selection vertex, we observe that if the assignment sets the corresponding literal to True, the selection vertex that is an out-neighbor of $\ell_{(j, \alpha)}$ must have been placed in a coalition that contains a palette vertex towards which $\ell_{(j, \alpha)}$ has positive utility, hence the utility of $\ell_{(j, \alpha)}$ is 2 and this vertex is stable.

For the converse direction, suppose that we have a Nash Stable partition π . We first prove that all vertices $u_{\lceil \frac{n}{d \log d} \rceil, i_2, 0}$, for $i_2 \in \{0, \dots, d-1\}$, must be in distinct coalitions. Indeed, if two of them are in the same coalition, p will have incentive to join the coalition that has the maximum number of such vertices. However, once p joins such a coalition, these vertices will have negative utility, contradicting stability. Second, we prove that for each i_1, i_2, j , the vertices $u_{(i_1, i_2, j)}, u_{(i_1, i_2, j+1)}$ must be in the same coalition. If not, consider two such vertices which are in distinct coalitions and maximize i_2 . We claim that in this case $c_{(i_1, j)}$ will always join $u_{(i_1, i_2, j)}$. Indeed, from the selection of i_2 , we have that for $i'_2 > i_2$, the contribution of arcs with absolute weight $4^{i'_2}$ to the utility of $c_{(i_1, j)}$ cancels out; while for $i'_2 < i_2$ the sum of all absolute utilities of arcs with weights $4^{i'_2}$ is too low to affect the placement of $c_{(i_1, j)}$ (in particular, $4^{i_2} - \sum_{j < i_2} 4^j > \sum_{j < i_2} 4^j$). But, if $c_{(i_1, j)}$ joins such a coalition, a selection vertex has negative utility, contradicting stability.

From the two properties above we can now extract an assignment to ϕ . For each selection vertex $u_{(i_1, i_2, j)}$, if this vertex is in the same coalition as $u_{(\lceil \frac{n}{d \log d} \rceil, i'_2, 0)}$, we give an assignment to the variables represented by $u_{(i_1, i_2, j)}$ as described, that is, we write i'_2 in binary and use one bit for each variable. Note that the choice of j here is irrelevant, as we have shown that thanks to the consistency vertices, for each i_1, i_2 , all vertices $u_{(i_1, i_2, j)}$ are in the same coalition.

If $u_{(i_1, i_2, j)}$ is not in the same coalition as any $u_{(\lceil \frac{n}{d \log d} \rceil, i'_2, 0)}$, we set its corresponding variables in an arbitrary way. To see that this assignment satisfies clause j , consider s_j , which, without loss of generality is placed with s'_j . If three of the vertices $\ell_{(j,1)}, \ell_{(j,2)}, \ell_{(j,3)}$ are in the same coalition as s_j , then s_j has negative utility, contradiction. Hence, one of these vertices, say $\ell_{(j,1)}$, is in another coalition. But then, since the neighbors of this vertex among vertices $u_{(\lceil \frac{n}{d \log d} \rceil, i_2, j)}$ are all in distinct coalitions, $\ell_{(j,1)}$ is in the same coalition with one such vertex and its out-neighbor selection vertex. But this means that we have extracted an assignment from the corresponding vertex and that this assignment sets the corresponding literal to True, satisfying the clause. ◀

▶ **Corollary 6.** *If the ETH is true, there is no algorithm which decides if an ASHG on a graph with n vertices, maximum degree Δ , and constant pathwidth admits a Nash Stable partition in time $\Delta^{o(\Delta)}(nW)^{O(1)}$, where W is the maximum absolute weight.*

Proof. We use the same reduction as in Theorem 5, from a 3-SAT formula on n variables, but set $d = \lfloor \frac{n}{2 \log n} \rfloor$. According to the properties of the construction, the pathwidth of the resulting graph is $O(\frac{n}{d \log d}) = O(1)$, the maximum degree is $O(n/\log n)$, the maximum weight is $2^{O(n/\log n)}$ and the size of the constructed graph is polynomial in n . If there exists an algorithm for finding a Nash Stable partition in the stated time, this gives a $2^{o(n)}$ algorithm for 3-SAT. ◀

▶ **Theorem 7.** *If the ETH is true, there is no algorithm which decides if an ASHG on a graph with n vertices, constant maximum degree Δ , and pathwidth p admits a Nash Stable partition in time $p^{o(p)}n^{O(1)}$, even if all weights have absolute value $O(1)$.*

▶ **Corollary 8.** *Theorem 7 also applies to CONNECTED NASH STABILITY.*

4 Parameterization by Treewidth Only

In this section we consider NASH STABILITY on graphs of bounded treewidth. Peters [36] showed that this problem is strongly NP-hard on stars, but for a more general version where preferences are described by boolean formulas (HC-nets). In Section 4.1 we strengthen this hardness result by showing that NASH STABILITY remains strongly NP-hard on stars for additive preferences. We also show that CONNECTED NASH STABILITY is strongly NP-hard on stars, albeit also using HC-nets.

The only case that remains is CONNECTED NASH STABILITY with additive preferences. Somewhat surprisingly, we show that this case evades our hardness results because it *is* in fact more tractable. We establish this via an algorithm running in pseudo-polynomial time when the treewidth is constant in Section 4.2. As a result, this is the only case of the problem which is not strongly NP-hard on bounded treewidth graphs (unless P=NP).

We then observe that our algorithm only establishes that the problem is in XP parameterized by treewidth (for weights written in unary). We show in Section 4.3 that this is inevitable, as the problem is W[1]-hard parameterized by treewidth even when weights are constant. Hence, our “pseudo-XP” algorithm is qualitatively optimal.

4.1 Refined paraNP-hardness

▶ **Theorem 9.** *NASH STABILITY of ASHG is strongly NP-hard for stars.*

Proof. We present a reduction from 3-PARTITION. In this problem we are given a set of $3n$ positive integers A , a target value T , and are asked to partition A into n triples, such that each triple has sum exactly T . This problem has long been known to be strongly

NP-hard [22]. Furthermore, we can assume that the sum of all elements of A is nT (otherwise the answer is clearly No); and that all elements have values strictly between $T/4$ and $T/2$, so sets of sizes other than three cannot have sum T (this can be achieved by adding T to all elements and setting $4T$ as the new target).

We construct an ASHG as follows: for each element of A we construct a vertex; we construct a set B of n additional vertices; we add a “stalker” vertex s and a helper s' . The preferences are defined as follows: for all $x \in A \cup B$ we set $w(x, s) = -1$; for each $x \in B$ we set $w(s, x) = 2T$; for each $x \in A$ we set $w(s, x) = -w(x)$, where $w(x)$ is the value of the corresponding element in the original instance. Finally, we set $w(s, s') = T$ and $w(s', s) = 1$. The graph is a star as all arcs are incident on s .

If there exists a valid 3-partition of A , we construct a stable partition of the new instance by placing s with s' and, for each triple placing its elements in a coalition with a distinct vertex of B . Vertices of $A \cup B$ have utility 0 in this configuration and no incentive to deviate; while s would have utility T in any existing coalition, so it has no incentive to leave s' ; s' is satisfied as she is together with s .

For the converse direction, if we have a stable configuration π , s' must be with s (otherwise s' has incentive to deviate). Furthermore, s cannot be with any vertex of $A \cup B$, as placing s with any such vertex would give that vertex incentive to leave. Hence, s, s' are one coalition of the stable partition, and s has utility T in this coalition. This implies that every coalition formed by vertices of $A \cup B$ must have utility at most T for s .

We now want to prove that every coalition of vertices of $A \cup B$ contains exactly one vertex of B . If we show this, then the weight of elements of A placed in each such coalition must be at least T , hence it must be exactly T (as the sum of all elements of A is nT). Therefore, we obtain a solution to the original instance.

To prove that every coalition that contains vertices of $A \cup B$ must contain exactly one vertex of B , suppose first that there exists a coalition that only contains vertices of A . Call the union of all such coalitions $A' \subseteq A$. Let C_1, \dots, C_k be the coalitions that contain some vertex of B , for some $k \leq |B| = n$. We now reach a contradiction as follows: first, since s does not have incentive to join C_i , for $i \in [k]$, we have $\sum_{v \in C_i} w(s, v) \leq T$, therefore $\sum_{i=1}^k \sum_{v \in C_i} w(s, v) \leq kT \leq nT$. On the other hand, $\sum_{i=1}^k \sum_{v \in C_i} w(s, v) \geq \sum_{v \in B} w(s, v) + \sum_{v \in A \setminus A'} w(s, v) > 2nT - nT = nT$, because if A' is non-empty $\sum_{v \in A \setminus A'} w(s, v) > -nT$. Hence we have a contradiction and from now on we suppose that every coalition that contains a vertex of $A \cup B$ has non-empty intersection with B .

Finally, consider a coalition that contains $k \geq 1$ vertices of B . These vertices give s utility $2kT$, meaning that the sum of weights of vertices of A placed in this coalition must be at least $(2k - 1)T$. Let t_i be the number of coalitions which contain exactly $i \geq 1$ vertices of B . We obtain the inequality $\sum_i t_i(2i - 1)T \leq nT$, because the weight of all elements of A is nT . On the other hand $\sum_i it_i = n$, as we have that $|B| = n$. We therefore have $\sum_i t_i(2i - 1) \leq n \Leftrightarrow \sum_i t_i \geq n = \sum_i it_i \Leftrightarrow \sum_{i>1} (1 - i)t_i \geq 0$, which can only hold if $t_i = 0$ for $i > 1$. \blacktriangleleft

► **Theorem 10.** *Deciding if a graphical hedonic game represented by an HC-net admits a connected Nash Stable partition is NP-hard even if the input graph is a star and all weights are in $\{1, -1\}$.*

4.2 Pseudo-XP algorithm for Connected Partitions

► **Theorem 11.** *There exists an algorithm which, given an ASHG instance on n vertices with maximum absolute weight W , along with a tree decomposition of the underlying graph of width t , decides if a connected Nash Stable partition exists in time $(nW)^{O(t^2)}$.*

Proof. Due to space constraints, we only sketch the proof. The algorithm uses standard DP techniques. In addition to connectivity information about which vertices of the bag are in the same connected component of the same coalition (which takes $t^{O(t)}$ to store in the DP table), we store for each vertex the utility it would have if it joined the coalition of each other vertex in the bag, and also the best coalition it has seen in the part of the graph that has already been processed. This gives $(nW)^t$ combinations per vertex in the bag, hence a DP table of the claimed size, and allows us to verify that all vertices are stable. The key property is that, since coalitions are connected, a coalition that has already been seen and does not contain any members in the bag is complete, in the sense that no further vertex can later be added to the coalition (as it would become disconnected). ◀

4.3 ETH-based lower bound for Connected Partitions

► **Theorem 12.** *If the ETH is true, deciding if an ASHG of pathwidth p admits a connected Nash Stable configuration cannot be done in time $f(p) \cdot n^{o(p/\log p)}$ for any computable function f , even if all weights are in $\{-1, 1\}$.*

By a slight modification of the previous proof we also obtain weak NP-hardness for the case where the input graph has vertex cover 2.

► **Corollary 13.** *It is weakly NP-hard to decide if an ASHG on a graph with vertex cover 2 admits a connected Nash Stable partition.*

5 Conclusions and Open Problems

Our results give strong evidence that the precise complexity of NASH STABILITY parameterized by $t + \Delta$ is in the order of $(t\Delta)^{O(t\Delta)}$. It would be interesting to verify if the same is true for CONNECTED NASH STABILITY, as this problem turned out to be slightly easier when parameterized only by treewidth, and is only covered by Corollary 8 for the case of bounded-degree graphs. Of course, it would also be worthwhile to investigate the fine-grained complexity of other notions of stability. In particular, versions which are complete for higher levels of the polynomial hierarchy [37] may well turn out to have double-exponential (or worse) complexity parameterized by treewidth [30, 31]. Finally, it would be worth to investigate precise complexity of other stability notions of hedonic games (e.g., individual stability and core stability), or other variants of hedonic games (e.g., fractional hedonic games and social distance games).

References

- 1 Alessandro Aloisio, Michele Flammini, and Cosimo Vinci. The impact of selfishness in hypergraph hedonic games. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1766–1773. AAAI Press, 2020. URL: <https://aaai.org/ojs/index.php/AAAI/article/view/5542>.
- 2 Haris Aziz, Florian Brandl, Felix Brandt, Paul Harrenstein, Martin Olsen, and Dominik Peters. Fractional hedonic games. *ACM Trans. Economics and Comput.*, 7(2):6:1–6:29, 2019. doi:10.1145/3327970.
- 3 Haris Aziz, Felix Brandt, and Hans Georg Seedig. Computing desirable partitions in additively separable hedonic games. *Artif. Intell.*, 195:316–334, 2013.

- 4 Haris Aziz and Rahul Savani. Hedonic games. In *Handbook of Computational Social Choice*, pages 356–376. Cambridge University Press, 2016.
- 5 Coralio Ballester. NP-completeness in hedonic games. *Games Econ. Behav.*, 49(1):1–30, 2004.
- 6 Nathanaël Barrot, Kazunori Ota, Yuko Sakurai, and Makoto Yokoo. Unknown agents in friends oriented hedonic games: Stability and complexity. 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 1756–1763. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33011756.
- 7 Nathanaël Barrot and Makoto Yokoo. Stable and envy-free partitions in hedonic games. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 67–73. ijcai.org, 2019. doi:10.24963/ijcai.2019/10.
- 8 Vittorio Bilò, Laurent Gourvès, and Jérôme Monnot. On a simple hedonic game with graph-restricted communication. In *SAGT*, volume 11801 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2019.
- 9 Niclas Boehmer and Edith Elkind. Individual-based stability in hedonic diversity games. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1822–1829. AAAI Press, 2020. URL: <https://aaai.org/ojs/index.php/AAAI/article/view/5549>.
- 10 Felix Brandt, Martin Bullinger, and Anaëlle Wilczynski. Reaching individually stable coalition structures in hedonic games. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 5211–5218. AAAI Press, 2021. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/16658>.
- 11 Simina Brânzei and Kate Larson. Coalitional affinity games and the stability gap. In *IJCAI*, pages 79–84, 2009.
- 12 Martin Bullinger and Stefan Kober. Loyalty in cardinal hedonic games. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 66–72. ijcai.org, 2021. doi:10.24963/ijcai.2021/10.
- 13 Katarína Cechlárová. Stable partition problem. In *Encyclopedia of Algorithms*, pages 2075–2078. Springer, 2016.
- 14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 15 Andreas Darmann, Edith Elkind, Sascha Kurz, Jérôme Lang, Joachim Schauer, and Gerhard J. Woeginger. Group activity selection problem with approval preferences. *Int. J. Game Theory*, 47(3):767–796, 2018. doi:10.1007/s00182-017-0596-4.
- 16 Vladimir G. Deineko and Gerhard J. Woeginger. Two hardness results for core stability in hedonic coalition formation games. *Discret. Appl. Math.*, 161(13-14):1837–1842, 2013.
- 17 Edith Elkind, Angelo Fanelli, and Michele Flammini. Price of pareto optimality in hedonic games. *Artif. Intell.*, 288:103357, 2020.
- 18 Edith Elkind and Michael J. Wooldridge. Hedonic coalition nets. In *AAMAS (1)*, pages 417–424. IFAAMAS, 2009.
- 19 Angelo Fanelli, Gianpiero Monaco, and Luca Moscardelli. Relaxed core stability in fractional hedonic games. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 182–188. ijcai.org, 2021. doi:10.24963/ijcai.2021/26.

- 20 Michele Flammini, Bojana Kodric, Gianpiero Monaco, and Qiang Zhang. Strategyproof mechanisms for additively separable and fractional hedonic games. *J. Artif. Intell. Res.*, 70:1253–1279, 2021.
- 21 Martin Gairing and Rahul Savani. Computing stable outcomes in symmetric additively separable hedonic games. *Math. Oper. Res.*, 44(3):1101–1121, 2019.
- 22 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 23 Tesshu Hanaka, Hironori Kiya, Yasuhide Maei, and Hirotaka Ono. Computational complexity of hedonic games on sparse graphs. In *PRIMA*, volume 11873 of *Lecture Notes in Computer Science*, pages 576–584. Springer, 2019.
- 24 Ararat Harutyunyan, Michael Lampis, and Nikolaos Melissinos. Digraph coloring and distance to acyclicity. In *STACS*, volume 187 of *LIPICs*, pages 41:1–41:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 25 Samuel Ieong and Yoav Shoham. Marginal contribution nets: a compact representation scheme for coalitional games. In *EC*, pages 193–202. ACM, 2005.
- 26 Ayumi Igarashi and Edith Elkind. Hedonic games with graph-restricted communication. In *AAMAS*, pages 242–250. ACM, 2016.
- 27 Ayumi Igarashi, Kazunori Ota, Yuko Sakurai, and Makoto Yokoo. Robustness against agent failure in hedonic games. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 364–370. ijcai.org, 2019. doi:10.24963/ijcai.2019/52.
- 28 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 29 Michael Lampis. Minimum stable cut and treewidth. In *ICALP*, volume 198 of *LIPICs*, pages 92:1–92:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 30 Michael Lampis, Stefan Mengel, and Valia Mitsou. QBF as an alternative to Courcelle’s theorem. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2018. doi:10.1007/978-3-319-94144-8_15.
- 31 Michael Lampis and Valia Mitsou. Treewidth with a quantifier alternation revisited. In *IPEC*, volume 89 of *LIPICs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
- 32 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018.
- 33 Kazunori Ohta, Nathanaël Barrot, Anisse Ismaili, Yuko Sakurai, and Makoto Yokoo. Core stability in hedonic games among friends and enemies: Impact of neutrals. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 359–365. ijcai.org, 2017. doi:10.24963/ijcai.2017/51.
- 34 Martin Olsen. Nash stability in additively separable hedonic games and community structures. *Theory Comput. Syst.*, 45(4):917–925, 2009.
- 35 Martin Olsen, Lars Bækgaard, and Torben Tambo. On non-trivial nash stable partitions in additive hedonic games with symmetric 0/1-utilities. *Inf. Process. Lett.*, 112(23):903–907, 2012.
- 36 Dominik Peters. Graphical hedonic games of bounded treewidth. In *AAAI*, pages 586–593. AAAI Press, 2016.
- 37 Dominik Peters. Precise complexity of the core in dichotomous and additive hedonic games. In *ADT*, volume 10576 of *Lecture Notes in Computer Science*, pages 214–227. Springer, 2017.

- 38 Dominik Peters and Edith Elkind. Simple causes of complexity in hedonic games. In *IJCAI*, pages 617–623. AAAI Press, 2015.
- 39 Walid Saad, Zhu Han, Tamer Basar, Mérouane Debbah, and Are Hjørungnes. Hedonic coalition formation for distributed task allocation among wireless agents. *IEEE Trans. Mob. Comput.*, 10(9):1327–1344, 2011. doi:10.1109/TMC.2010.242.
- 40 Jakub Sliwinski and Yair Zick. Learning hedonic games. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 2730–2736. ijcai.org, 2017. doi:10.24963/ijcai.2017/380.
- 41 Shao Chin Sung and Dinko Dimitrov. Computational complexity in additive hedonic games. *Eur. J. Oper. Res.*, 203(3):635–639, 2010.
- 42 Gerhard J. Woeginger. A hardness result for core stability in additive hedonic games. *Math. Soc. Sci.*, 65(2):101–104, 2013.

Fine-Grained Complexity Lower Bounds for Families of Dynamic Graphs

Monika Henzinger ✉

Department of Computer Science, Universität Wien, Austria

Ami Paz ✉

LISN, CNRS & Paris-Saclay University, France

A. R. Sricharan ✉

Department of Computer Science, Universität Wien, Austria

Abstract

A dynamic graph algorithm is a data structure that answers queries about a property of the current graph while supporting graph modifications such as edge insertions and deletions. Prior work has shown strong conditional lower bounds for general dynamic graphs, yet graph families that arise in practice often exhibit structural properties that the existing lower bound constructions do not possess. We study three specific graph families that are ubiquitous, namely constant-degree graphs, power-law graphs, and expander graphs, and give the first conditional lower bounds for them. Our results show that even when restricting our attention to one of these graph classes, any algorithm for fundamental graph problems such as distance computation or approximation or maximum matching, cannot simultaneously achieve a sub-polynomial update time and query time. For example, we show that the same lower bounds as for general graphs hold for *maximum matching* and *(s, t)-distance* in constant-degree graphs, power-law graphs or expanders. Namely, in an m -edge graph, there exists no dynamic algorithms with both $O(m^{1/2-\epsilon})$ update time and $O(m^{1-\epsilon})$ query time, for any small $\epsilon > 0$. Note that for *(s, t)-distance* the trivial dynamic algorithm achieves an almost matching upper bound of constant update time and $O(m)$ query time. We prove similar bounds for the other graph families and for other fundamental problems such as densest subgraph detection and perfect matching.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic graph algorithms, Expander graphs, Power-law graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.65

Related Version *Full Version:* <https://arxiv.org/abs/2208.07572>

Funding *Monika Henzinger and A. R. Sricharan:* 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. 101019564 “The Design of Modern Fully Dynamic Data Structures (MoDynStruct)” and from the Austrian Science Fund (FWF) project “Fast Algorithms for a Reactive Network Layer (ReactNet)”, P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024



1 Introduction

A dynamic graph algorithm is a data structure that stores a graph and supports update operations, usually consisting of edge insertions and deletions, as well as query operations that ask about a specific property of the graph. The introduction of strong conditional lower bounds based on widely-believed complexity assumptions [1, 12] has had a fundamental influence on the field, pushing the design of new algorithms towards more specialized algorithms such as partially-dynamic or even offline-dynamic algorithms or towards approximate solutions. However, graphs arising in real-world applications often differ significantly from the very specifically crafted graphs for which the lower bound results are shown. Frequently, real-world



© Monika Henzinger, Ami Paz, and A. R. Sricharan;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, 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

graphs have some special structure, such as having a power-law degree distribution, a constant degree, or being planar. Expanders, on the other side, have recently been used to design dynamic algorithms for *general* graphs. This naturally leads to the question of determining the complexity of dynamic graph algorithms for these graph classes, and this is exactly the question investigated in this paper.

While the complexity of dynamic graph algorithms for planar graphs has already been studied quite extensively [20, 14, 18, 16, 3, 2, 1, 15, 5], the question is still widely open for other families of graphs, including power-law graphs, constant-degree graphs, and expanders. Certain problems become easier for these graph classes: As an N -node¹ constant-degree graph has $O(N)$ edges, computing all-pairs shortest paths (APSP) takes only time $\tilde{O}(N^2)$, while the popular APSP conjecture postulates that for general graphs, there exists a small constant $c > 0$ such that any algorithm in the word RAM model with $O(\log N)$ -bit words requires $N^{3-o(1)}$ expected time to compute APSP. Moreover, some problems become trivial in these graph classes, e.g., computing shortest paths with logarithmic additive error on expander graphs is trivial, due to their low diameter.

In this paper we will concentrate on graph problems that have real-world applications such as shortest-paths (which has applications in online navigation), matching (which has applications in reconfigurable datacenters), and densest subgraphs (which has applications in network analysis), yet we believe that our general approach can be applied to further graph problems. For these three problems, the known conditional lower bounds construct graphs that are far from being in the classes we consider: They have maximum degree $\Omega(N)$ and small cuts, and their degree distribution is unknown as it depends on the instance that is postulated to be hard.

Constant-degree graphs. Various dynamic graph problems that admit strong lower bounds in general graphs have very efficient algorithms on constant-degree graphs. Let Δ be the maximum degree in the graph. For *local* problems, where the solution at a node v can be computed by simply analyzing information stored at the neighbors of v such as maintaining a maximal matching, a maximal independent set, or a $(\Delta + 1)$ -vertex coloring, there exist simple dynamic algorithms with $O(\Delta)$ update time and constant query time. Additionally, for various problems that count or detect certain fixed subgraphs with c nodes (such as triangle counting for $c = 3$) there exists dynamic algorithms with $O(\Delta^{c-1})$ update time and constant query time, even though they have polynomial conditional lower bounds in general graphs (see Table 1). These efficient algorithms for local problems rule out the possibility of any non-trivial lower bound in the constant-degree setting.

Furthermore, even for the non-local problem of maintaining a maximum matching Gupta and Peng [11] designed a $(1 + \delta)$ -approximation algorithm for any small $\delta > 0$ that runs in $O\left(\min\left\{\frac{m}{|M(t)|}, |M(t)|\right\}\delta^{-2}\right)$ amortized time per update, where $M(t)$ denotes the maximum cardinality matching after the t -th update operation. As in a graph with maximum degree Δ it holds that $|M(t)| \geq m/(2\Delta)$, their algorithm achieves an amortized update time of $O(\Delta\delta^{-2})$, which is $O(\delta^{-2})$ in constant-degree graphs. This raises the question of how efficiently other non-local dynamic graph problems such exact maximum matching, shortest paths, and densest subgraph can be solved in dynamic constant-degree graphs and whether it is possible to show (conditional) lower bounds for them.

¹ To avoid confusion with the parameters n and M used in the online-matrix-vector multiplication conjecture, we use N to denote the number of vertices and m the number of edges in the dynamic graphs.

■ **Table 1** Counting problems which admit polynomial conditional lower bounds on general graphs (amortized) and on Erdős-Rényi graphs (average case), but have algorithms with constant update and query times in constant-degree graphs. For the lower bounds above, there is no dynamic algorithm with pre-processing time $p(m, N)$, update time $u(m, N)$, and query time $q(m, N)$ unless the OMv conjecture is false. When $p(m, N)$ is unspecified, $\text{poly}(N)$ pre-processing time is allowed.

Problems	Lower bounds					Upper bounds	
	General graphs [12]		Erdős-Rényi avg-case [13]			constant Δ (trivial)	
	$u(m, N)$	$q(m, N)$	$p(m, N)$	$u(m, N)$	$q(m, N)$	$u(m, N)$	$q(m, N)$
Triangle counting	$m^{1/2-\epsilon}$	$m^{1-\epsilon}$	$N^{3-\epsilon}$	$m^{1/2-\epsilon}$	$m^{1-\epsilon}$	Δ	1
C_4 subgraph counting						Δ^3	1
5-length (s, t) -path count			$N^{2-\epsilon}$	$N^{\omega-2-\epsilon}$	1	Δ^4	1

Expanders. Expander decompositions are increasingly becoming a central tool for designing dynamic graph algorithms with improved running time bounds for various graph problems such as connectivity, minimum spanning tree, shortest paths, conductance, edge-connectivity, maximum flows, and minimum cuts [17, 9, 7, 6]. One of the central subproblems that these algorithms have to handle is to solve a graph problem on a dynamically changing expander. To understand the limitations of this approach it is crucial to understand which problems can be solved efficiently on expanders, and which cannot. We present novel lower bounds for dynamic problems on expanders, more specifically on constant-degree expanders.

Note that these results also have an interesting connection to the average-case hardness of dynamic graph algorithms. Recently, lower bounds on the average-case hardness were shown for various subgraph counting problems in dynamic Erdős-Rényi graphs (see Table 1 for some of them) [13]. As random graphs are usually expanders, giving lower bounds for a problem on dynamic expanders gives an indication that this problem might also be hard in the average case and can motivate further work in this direction.

Power-law graphs. Graphs are called *power-law graphs* if the fraction of nodes with degree d is proportional to $1/d^c$ for some constant $c > 0$. Static and dynamic power-law graphs arise surprisingly often in real-world networks, such as the Internet, the world-wide web, and citation graphs, as well as in physics, linguistics, and economics. Even though the existence of large dynamic power-law graphs was already pointed out in 2004 [10], no efficient dynamic algorithms have been developed specifically for this class of graphs. This leads to the question of whether sub-polynomial time dynamic algorithms are even possible for power-law graphs or not. In fact, dynamic power-law graphs were not only never studied, they were not even defined – removing even a single edge from a power-law graph changes the degree distribution and thus violates the power-law distribution. Hence, we first present several definitions of *dynamic* power-law graphs, where some slackness in the degree-distribution is allowed. Then we prove lower bounds that hold for all of these dynamic power-law graph definitions.

1.1 Our Results

Throughout the paper we use the standard assumption that queries output one value, such as the size, length or weight of the solution. Note that this makes it only more challenging to prove lower bounds. All our results are conditioned on the popular OMv conjecture [12], defined in Section 2, but to simplify the terminology we usually drop the word “conditional”. Our results are also summarized in Table 2.

1. *Main results.* We study the hardness of dynamic algorithms for (i) constant-degree graphs, (ii) expanders, and (iii) power-law graphs, for the following graph problems: Determining (a) the size of a maximum matching, (b) the length of the (s, t) shortest-path (i.e. (s, t) -distance), and (c) the density of the densest subgraph. Specifically, we show the following tradeoff between the update time u and the query time q in an m -edge graph for maximum matching and (s, t) -distance: There is no dynamic algorithm which achieves both $u = O(m^{1/2-\epsilon})$ and $q = O(m^{1-\epsilon})$ for any small $\epsilon > 0$. Note that these bounds match the bounds given for general graphs in [12] and that the lower bound for (s, t) -distance is almost tight as the simple algorithm that only records the edge change at update time and computes the solution from scratch at query time achieves $u = O(1)$ and $q = O(m)$. For densest subgraph we show that there is no dynamic algorithm which achieves both $u = O(N^{1/4-\epsilon})$ and $q = O(N^{1/2-\epsilon})$ for any small $\epsilon > 0$, which is weaker than the lower bound on general graphs (of $u = O(N^{1/2-\epsilon})$ and $q = O(N^{1-\epsilon})$).

The only relevant prior work are conditional lower bounds for planar graphs [1], which have constant degree: In unweighted graphs they show for all-pairs-shortest paths a weaker tradeoff between update time u and query time q than we do, namely they prove $\max(u^2 \cdot q, u \cdot q^2) = \Omega(m^{1-o(1)})$. In *weighted* graphs they show for (s, t) -distance a tradeoff of $\max(u, q) = \Omega(m^{1/2-o(1)})$. Note that our result is stronger as it shows that in *unweighted* graphs no algorithm with $u = \Omega(m^{49/100})$ and $q = \Omega(m^{99/100})$ is possible.

2. *Degree-lower bound trade-off.* While the constant-degree lower bounds are equal to the lower bounds for general graphs in terms of m , they are naturally quadratically lower in terms of the number of nodes N . To understand the behaviour of the bounds also with respect to N , we extend our constant-degree lower bounds for maximum matching, perfect matching, and (s, t) -distance to graphs with maximum degree $O(N^t)$, for any $0 \leq t \leq 1$. We show the following result: There is no dynamic algorithm which achieves both $u = O(N^{(1+t)/2-\epsilon})$ and $q = O(N^{1+t-\epsilon})$ in a graph with maximum degree $O(N^t)$ for any $\epsilon > 0$. These results hold even in bipartite graphs. Note that for $t = 1$ these results match exactly the bounds for general graphs in [12], and for $t = 0$, they match the aforementioned results for constant-degree graphs.
3. *Approximation results.* In constant-degree graphs we extend the lower bound to the problem of approximating the (s, t) -distance within a factor of $3 - \delta$, for any small constant δ . This naturally extends the $(3 - \delta)$ -approximation lower bounds on general graphs to the constant-degree case. In planar graphs, the (s, t) -distance lower bound holds only for exact answers.

Note that a similar extension to approximation algorithms is not possible for maximum cardinality matching and for densest subgraph: (a) For maximum matching, for any small $\delta > 0$ the above-mentioned $(1 + \delta)$ -approximation algorithm [11] achieves an amortized update time of $O(\delta^{-2})$, which is constant for a constant δ , thereby precluding any non-trivial lower bounds for approximate maximum matching in the constant-degree setting. Stated differently, *our work shows an interesting dichotomy for dynamic matching in constant-degree graphs*: For the exact setting there is no dynamic algorithm which achieves both $u = O(m^{1/2-\epsilon})$ and $q = O(m^{1-\epsilon})$ for any small $\epsilon > 0$, while a $(1 + \delta)$ -approximation can be achieved in constant time, for any small $\delta > 0$. (b) *The same dichotomy arises for densest subgraph*: For any small $\delta > 0$ there exists a $(1 - \delta)$ -approximation algorithm with polylogarithmic time per operation [19], while we show a polynomial lower bound for the exact value.

4. *Partially dynamic algorithms.* We extend the constant-degree reductions for maximum matching and (s, t) -distance also to the insertions-only and the deletions-only setting, achieving the same lower bound as in the fully dynamic setting.

■ **Table 2** Our results for graphs on N nodes with m edges. For every u and q stated above, there is no algorithm for the corresponding problem with amortized $O(u(m, N))$ update time and $O(q(m, N))$ query time simultaneously unless the OMv conjecture is false. The first three rows hold also for perfect matching. All the lower bounds in the table except for densest subgraph match the general OMv lower bounds.

Problem	Class	$u(m, N)$	$q(m, N)$
Maximum Matching	$\Delta \leq 3$	$m^{1/2-\epsilon}$	$m^{1-\epsilon}$
	constant degree & expansion		
	power-law graphs		
	$\Delta \leq 3$, partially dynamic		
	$\Delta \leq N^t$	$N^{(1+t)/2-\epsilon}$	$N^{1+t-\epsilon}$
(s, t) -distance	$\Delta \leq 3$	$m^{1/2-\epsilon}$	$m^{1-\epsilon}$
	$(3 - \delta)$ -approx, $\Delta \leq 3$		
	constant degree & expansion		
	power-law graphs		
	$\Delta \leq 3$, partially dynamic		
	$\Delta \leq N^t$	$N^{(1+t)/2-\epsilon}$	$N^{1+t-\epsilon}$
Densest Subgraph	$\Delta \leq 5$	$N^{1/4-\epsilon}$	$N^{1/2-\epsilon}$
	constant degree & expansion		
	power-law graphs		

5. *Perfect matching.* A special case of maximum cardinality matching is determining whether a perfect matching exists in a bipartite graph. For constant-degree graphs and expander graphs we show the following lower bound: There is no dynamic algorithm which achieves both $u = O(m^{1/2-\epsilon})$ and $q = O(m^{1-\epsilon})$ for any small $\epsilon > 0$. This can also be extended to the varying-degree setting.

To summarize, our paper opens up the research field of dynamic graph algorithms for more specific, practical graph classes, in contrast with previous work that concentrated on general or planar graphs. We believe that our techniques can be extended to further classes of dynamic graphs or even in other domains of theoretical computer science, such as distributed graph algorithms or streaming algorithms. One further interesting implication of our work is presenting the limitations of dynamic graph algorithms on expanders, thus complementing recent algorithmic results that use expander decompositions in dynamic graphs.

1.2 Our Techniques

We prove lower bounds by reductions from the online matrix vector (OMv) conjecture [12]. In these reductions, the input of an online problem, which is an $n \times n$ matrix M and a sequence of n pairs (u, v) of n -vectors, is translated into a dynamic graph. The reduction is built so that there exists a pair (u, v) satisfying $uMv = 1$ if and only if the dynamic graph has some desired property at some point of time. While we follow the general framework of OMv lower bounds, the details are delicate, as the dynamic graphs we construct should fall into specific graph classes at all times, while still maintaining the graph property under consideration. We give a high-level overview of our reductions below.

One way to turn known OMv-to-dynamic graphs reductions into reductions that produce bounded-degree graphs is by replacing high-degree nodes by bounded-degree trees. This technique has a rather clear and straightforward effect on the distances in the graph, so it is applicable when considering distance-related problems. This, however, is far from being

the case when considering other problems, such as maximum matchings. Here, replacing a high-degree node with a gadget could adversely affect the desired matching size, since the gadget might create several augmenting paths that would not have existed when it was a single high-degree node. To overcome this, we limit the possible maximum matching sizes, by designing a reduction graph with bounded-degree gadgets composed of paths, where the maximum matching is always either a perfect matching, or a near-perfect matching, i.e., the matching size is either $N/2$ or $N/2 - 1$. This reduction thus involves a large matching and a small gap between the $uMv = 0$ and $uMv = 1$ cases, and hence cannot be extended to achieve a lower bound for the approximation of the maximum matching size. While this might seem as a limitation of our construction, recall that this is actually not the case: As described above, for any small $\delta > 0$ there is a constant time $(1 + \delta)$ -approximation dynamic algorithm for the problem, and, thus, such a lower bound cannot exist.

An even more delicate reduction we present is for proving a lower bound on the densest-subgraph problem. A straightforward reduction would change $O(n)$ graph-edges for every bit of the input, which will allow us to make sure that the density of the densest subgraph changes by a significant amount when $uMv = 0$ versus when $uMv = 1$. However, this would involve $O(n^2)$ updates for each (u, v) input pair, and the reduction would fail to yield any non-trivial lower bound. Thus, we are forced to change very few edges for each input bit, which renders an almost negligible effect on the density, making it difficult to control the exact density of the densest subgraph. Our reduction balances these two factors, using a construction where each gadget is a sufficiently dense regular graph, while having each bit of the input translate into the existence or nonexistence of merely two edges inside specific gadgets. As in the case of matchings, our lower bounds cannot be extended to approximations, as for any $\delta > 0$ there exists a fast algorithm with polylogarithmic update time for computing $(1 - \delta)$ -approximations to the densest subgraph.

We then extend these reductions from bounded-degree graphs to constant-degree *constant-expansion* graphs. First, the standard lower bound reductions contain sparse cuts if the inputs M, u or v are sparse, making a standard reduction graph far from being an expander. Thus, we have to augment the graph with many more edges to make sure that it has no sparse cuts regardless of M, u and v . We do this augmentation “inside a layer” to prevent the additional edges from creating undesired short paths between s and t , or spurious augmenting paths in the case of matchings. Sparse cuts also exist in parts of the graph that do not depend on M, u or v , and to handle these, we add edges of a constant-degree expander between a well-chosen set of nodes, thus guaranteeing the expansion without changing the required graph property. Finally, in the case of distance-related problems, we note that expander graphs can have at most logarithmic diameter, but the substitution of nodes by trees described above increases the diameter to be at least logarithmic, leaving only a very small slack for our construction.

When studying densest subgraphs on expanders, adding edges in order to avoid sparse cuts might change the location and structure of the densest subgraph in an undesired way. In order to guarantee the expansion in this case, we add a copy of all the graph nodes, build a constant-degree expander on the copies of the nodes, and then connect each node to its copy by a matching.

In dynamic power-law graphs where the node degrees may depend on the inputs u, M, v and change over time, we have to guarantee that the degree changes incurred by the processing of different inputs do not cause a violation of the power-law distribution of degrees. As before, all the changes must also be done without changing the graph property under consideration, and without performing too many update operations. We address this problem by inserting or deleting edges in an online fashion in other parts of the reduction graph, to compensate for the changes incurred by processing the input vector pairs.

Organisation. Section 2 has notation and definitions. Section 3 presents the dynamic maximum matching lower bounds. The lower bounds for other problems, namely densest subgraph detection and (s, t) -distance, are briefly discussed in Section 4, and so is the partially dynamic setting. The full lower bounds, the results for partially dynamic graphs, and the missing proofs all appear in the full version of the paper.

2 Preliminaries

Throughout the paper, we consider vectors and matrices that are boolean, and so a vector-matrix-vector multiplication outputs a single bit. Henzinger *et al.* [12] define the *Online Matrix Vector* (OMv) and the *Online Vector Matrix Vector* (OuMv) multiplication problems.

► **Definition 1** (Online Matrix Vector Multiplication). *Let M be a boolean $n \times n$ matrix. Preprocessing the matrix is allowed. Then, n vectors v^1, v^2, \dots, v^n arrive one at a time, and the task is to output the product Mv^i before the next vector is revealed.*

► **Definition 2** (Online Vector Matrix Vector Multiplication). *Let M be a boolean $n \times n$ matrix. Preprocessing the matrix is allowed. Then, n vector pairs $(u^1, v^1), (u^2, v^2), \dots, (u^n, v^n)$ arrive one at a time, and the task is to output the bit $u^i M v^i$ before the next vector pair is revealed.*

In their paper, they show that the OuMv problem can be reduced to the OMv problem, and conjecture that there is no truly subcubic time algorithm for OMv.

► **Conjecture 3** (OMv). *There is no algorithm for the OMv (and hence the OuMv) problem running in time $O(n^{3-\epsilon})$ for any constant $\epsilon > 0$.*

We work with the OuMv problem for all the reductions in our paper. We denote the length of our input vectors u^i, v^i by n , and thus the matrix M is of dimension $n \times n$. We use upper indices to indicate the vector's location in the stream, but usually focus on one pair (u, v) omitting these indices. We use lower indices for a location in the vector or matrix, e.g., u_i and M_{ij} . We use N to denote the number of nodes in our reduction graph.

► **Definition 4** (Expansion). *The expansion parameter of a graph $G = (V, E)$ is defined as*

$$h = \min \left\{ \frac{|E(S, \bar{S})|}{|S|} \mid \emptyset \neq S \subseteq V, |S| \leq |V|/2 \right\}$$

where $E(S, \bar{S})$ is the number of edges from S to $V \setminus S$. We call a graph with expansion h a *h -expander*. Works on dynamic algorithms use a different definition of expansion parameter h' , called *volume expansion*. However, when considering constant-degree graphs with constant expansion (as we do in this paper), both parameters are within a Δ factor of each other, so we only consider the expansion parameter h in our proofs.

We study *power-law* graphs as introduced in [4], and only consider the setting where $\beta > 2$. In the following definition, if the number of nodes N in the graph is fixed, then we get that α is roughly $\ln N$.

► **Definition 5** (Power Law). *A graph is said to follow an (α, β) -power law distribution if the number N_d of nodes with degree d is inversely proportional to d^β for some constant $\beta > 0$. That is,*

$$N_d = \left\lfloor \frac{e^\alpha}{d^\beta} \right\rfloor \approx \left\lfloor \frac{1}{\zeta(\beta)} \cdot \frac{N}{d^\beta} \right\rfloor,$$

where $\zeta(\beta) = \sum_{i=1}^{\infty} 1/i^\beta$ is the Riemann Zeta function.

Since dynamic graphs allow edge insertions and deletions, it is impossible to maintain an exact degree distribution at all times. Hence, we introduce the notion of approximate power-law distributions to afford some slack for dynamic changes. One natural relaxation is to allow β to vary within an interval.

► **Definition 6** (*β -Varying Power Law*). *A graph is said to follow an $(\alpha, \beta_1, \beta_2)$ -varying power law distribution if the number N_d of nodes with degree d satisfies*

$$\min \left\{ \left\lfloor \frac{1}{\zeta(\beta_1)} \cdot \frac{N}{d^{\beta_1}} \right\rfloor, \left\lfloor \frac{1}{\zeta(\beta_2)} \cdot \frac{N}{d^{\beta_2}} \right\rfloor \right\} \leq N_d \leq \max \left\{ \left\lceil \frac{1}{\zeta(\beta_1)} \cdot \frac{N}{d^{\beta_1}} \right\rceil, \left\lceil \frac{1}{\zeta(\beta_2)} \cdot \frac{N}{d^{\beta_2}} \right\rceil \right\},$$

This relaxation of an exact power law, while being natural, is a global relaxation rather than a local one. Thus we also define two locally approximate definitions below that allow similar slack for all degrees.

► **Definition 7** (*Additively Approximate Power Law*). *A graph is said to follow an (α, β, c) -additively approximate power law distribution if the number N_d of nodes of degree d for a realisable degree d satisfies*

$$\left\lfloor \frac{1}{\zeta(\beta)} \cdot \frac{N}{d^\beta} \right\rfloor - c \leq N_d \leq \left\lceil \frac{1}{\zeta(\beta)} \cdot \frac{N}{d^\beta} \right\rceil + c$$

where we say that d is a realisable degree if there is a node of degree d in an (α, β) -power law graph.

► **Definition 8** (*Multiplicatively Approximate Power Law*). *A graph is said to follow an $(\alpha, \beta, \epsilon)$ -multiplicatively approximate power law distribution if the number N_d of nodes of degree d satisfies*

$$\frac{1}{(1 + \epsilon)} \cdot \left\lfloor \frac{1}{\zeta(\beta)} \cdot \frac{N}{d^\beta} \right\rfloor \leq N_d \leq (1 + \epsilon) \cdot \left\lceil \frac{1}{\zeta(\beta)} \cdot \frac{N}{d^\beta} \right\rceil$$

Our lower bounds contain at most four nodes that are one degree away from an exact power-law distribution, and thus hold in all the models discussed above with any reasonable parameter regime.

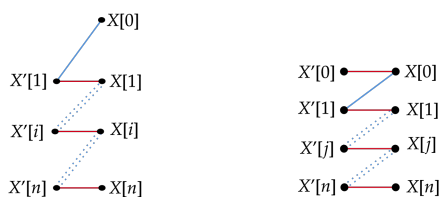
3 Lower Bounds for Dynamic Maximum Matching

The previous matching lower bounds on general graphs [12, 8] use reduction graphs that contain nodes with degree $\Omega(N)$. In this section, we construct a constant-degree reduction graph with constant expansion.

Reduction gadgets

Our gadget construction starts by replacing each node of a dense reduction by a path; we refer to each path as a “subgadget”.

Connecting every node of this new subgadget with nodes outside the subgadget might create unwanted matchings of larger sizes, so instead we carefully choose a subset of the path nodes to connect outside the subgadget. Figure 1 shows odd and even paths (“odd” and “even” describe the number of nodes) with a “canonical” matching for each of them marked in red. Next, we detail the connections outside the subgadgets.



■ **Figure 1** Odd and even sized paths used in the maximum matching lower bounds. The canonical matchings are marked in red.

Consider an odd path on $2n + 1$ vertices, and a bipartition of the vertices into (X', X) with $|X'| \leq |X|$. Indexing the vertices as $X[0]$ and $X'[i], X[i]$ for $1 \leq i \leq n$, our canonical matching matches $X[i]$ with $X'[i]$, and leaves $X[0]$ unmatched. We connect only the vertices $\{X[i] \mid 1 \leq i \leq n\}$ outside the subgadget, while vertices in X' and $X[0]$ only have edges inside the subgadget. For an even path on $2n + 2$ vertices indexed as above, our canonical matching is perfect, and matches $X[i]$ to $X'[i]$. Only vertex $X'[0]$ and all the vertices in X are connected outside the subgadget, and all vertices $X'[i], 1 \leq i \leq n$, only have edges within the subgadget.

While this suffices for sparsification, we need additional constructions in order to guarantee constant expansion. In particular, it turns out that adding edges inside a subgadget does not suffice for constant expansion, and we are forced to add edges between subgadgets. Our construction adds edges on the same side of the bipartition across subgadgets, and we show that if the newly added edges take part in any augmenting path, then there also exists an augmenting path in the subgraph devoid of any newly inserted edge.

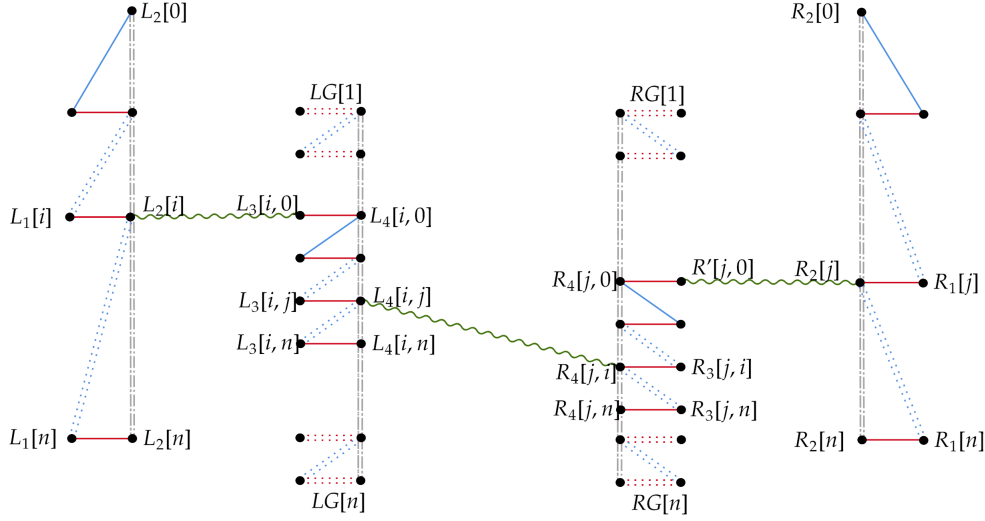
The reduction graph consists of a left subgraph (L) and a right subgraph (R), connected together by edges corresponding to the matrix M . Note that for constant expansion, we need the number of edges of M to be a constant fraction of the sizes of L and R . While it would be possible to construct a reduction graph with $|L|$ and $|R|$ that depend on the input matrix M , we instead choose to augment the input matrix and vectors as it simplifies notation. We thus augment the input beforehand to ensure that there are $\Omega(n^2)$ edges crossing from L to R . To this end, we work with the vectors $\hat{u} = (u \ 0)$ and $\hat{v} = (v \ 0)$ of dimension $2n$, and the matrix $\hat{M} = \begin{pmatrix} M & \\ & 1 \end{pmatrix}$ of dimension $2n \times 2n$. It is easy to see that $uMv = 1 \iff \hat{u}\hat{M}\hat{v} = 1$.

► **Definition 9 (Reinforced gadget).** A reinforced gadget with x subgadgets of size y consists of x subgraphs, each of which is a path on y nodes. The nodes are bipartitioned into sets (X', X) with the larger side of the partition labelled as X in each subgadget. Thus $|X'| \leq |X|$. It is then augmented with the following edge-set: Consider a degree- d expander graph on $x \cdot \lceil \frac{y}{2} \rceil$ nodes, choose an arbitrary bijection between the expander nodes and X , and add the expander edges to these nodes accordingly. The resulting graph is the reinforced gadget.

Note that reinforced gadgets are not bipartite. Thus, while the constant-degree lower bounds hold for bipartite matching, the expander result is for maximum matching on general graphs. In what follows, we drop the hats from \hat{u}, \hat{M} and \hat{v} for simplicity, but continue our analysis with their dimensions as $2n, 2n \times 2n, 2n$ respectively.

Reduction Graph

We use the following reduction graph, composed of two odd-sized reinforced gadgets and two even-sized reinforced gadgets.



■ **Figure 2** The expander reduction graph for maximum matching. The red lines denote the canonical matching, the blue lines denote the paths in each subgadget, the grey lines denote the expander edges, and the green lines denote the input-dependent edges.

- A reinforced gadget with one subgadget of size $4n + 1$, on a set $L_1 \cup L_2$. The nodes are labelled $L_1[i]$ for $1 \leq i \leq 2n$, and $L_2[i]$ for $0 \leq i \leq 2n$. The path is from $L_2[0]$ to $L_2[2n]$, and the expander is on L_2 .
- A reinforced gadget with $2n$ subgadgets of size $4n + 2$, on a set $L_3 \cup L_4$. The subgadgets are labelled $LG[i]$ for $1 \leq i \leq 2n$, and the nodes of subgadget $LG[i]$ are labelled $L_3[i, j]$ or $L_4[i, j]$ for $0 \leq j \leq 2n$ depending on whether the node is in L_3 or L_4 . The path in each subgadget goes from $L_3[i, 0]$ to $L_4[i, 2n]$, and the expander is on L_4 .
- A copy of the above structure, with node sets marked R_i instead of L_i , respectively.
- For the matrix M , add the edge $(L_4[i, j], R_4[j, i])$ if $M_{ij} = 1$.
- Given an input vector u , for each $i \in [2n]$, add the edge $(L_2[i], L_3[i, 0])$ if $u_i = 1$, and $(L_2[i], L_4[i, 0])$ otherwise.
- Given an input vector v , for each $j \in [2n]$, add the edge $(R_2[j], R_3[j, 0])$ if $v_j = 1$, and add the edge $(R_2[j], R_4[j, 0])$ otherwise.

The total number of nodes in the reduction graph is $N = 16n^2 + 16n + 2 = \Theta(n^2)$.

Matchings in the Graph

We start by defining a base matching B on the graph, which is made up of the canonical matchings on each of the gadgets. On the left side, B matches $L_3[i, j]$ to $L_4[i, j]$, and $L_1[i]$ to $L_2[i]$ for all i, j . The matching on the right side is similar. Note that this matching always exists regardless of the input, and only $L_2[0]$ and $R_2[0]$ are unmatched in the entire graph. Thus $|B| = \frac{N}{2} - 1$. We claim that this graph has a perfect matching if and only if $uMv = 1$. Let C denote the maximum cardinality matching.

▶ **Lemma 10.** *If $uMv = 1$, then $|C| = \frac{N}{2}$, and otherwise $|C| = \frac{N}{2} - 1$.*

Proof. Since B is always a matching of size $\frac{N}{2} - 1$ regardless of the input, the claim is equivalent to showing that $uMv = 1$ if and only if there is an augmenting path with respect to the matching B .

(\implies) Suppose that $uMv = 1$, with $u_i = M_{ij} = v_j = 1$. Consider the path P composed of the following subpaths, of which all except P_4 start with an unmatched edge and end with a matched edge, while P_4 both starts and ends with an unmatched edge.

- $P_1 = L_2[0], L_1[1], L_2[1], \dots, L_2[i]$
- $P_2 = L_2[i], L_3[i, 0], L_4[i, 0], \dots, L_4[i, j]$
- $P_3 = L_4[i, j], R_4[j, i], R_3[j, i], \dots, R_3[j, 0]$
- $P_4 = R_3[j, 0], R_2[j], R_1[j], \dots, R_2[0]$

Thus, P is an augmenting path to the base matching B , which gives us that the maximum matching C has to have size $> \frac{N}{2} - 1$, implying that the maximum matching C is a perfect matching.

(\impliedby) Suppose now that there exists an augmenting path P to the base matching B that starts at $s = L_2[0]$ and ends at $t = R_2[0]$.

- Since $(\cup_i L_i, \cup_j R_j)$ is an (s, t) -cut, there is at least one crossing edge, say $(L_4[i, j], R_4[j, i])$, in P . Thus $M_{ij} = 1$.
- Since P leaves the subgadget $LG[i]$ using $(L_4[i, j], R_4[j, i])$, it should have entered the subgadget at some previous instance. Since $(L_4[i, j], R_4[j, i])$ is an unmatched edge and all the matching edges in $LG[i]$ are within the subgadget, P should have entered the subgadget using an unmatched edge. As all the matching edges in $LG[i]$ are between L_3 and L_4 , P cannot both enter and exit the subgadget through L_4 . Thus P enters $LG[i]$ through L_3 . However, the only possible unmatched edge from L_3 leaving the subgadget is the edge $(L_3[i, 0], L_2[i])$. Thus P uses the edge $(L_3[i, 0], L_2[i])$ to enter the subgadget $LG[i]$, and so $u_i = 1$.
- The path P now enters the subgadget $RG[j]$ through the unmatched edge $(L_4[i, j], R_4[j, i])$. As before, all the matched edges in $RG[j]$ are between R_4 and R_3 , and so P has to exit the subgadget using an unmatched edge from R_3 . However, the only possible unmatched edge from R_3 leaving the subgadget is the edge $(R_3[j, 0], R_2[j])$. Thus the edge $(R_3[j, 0], R_2[j])$ is used by P , giving us that $v_j = 1$.

This gives us that $uMv = 1$ as required. \blacktriangleleft

Complexity of the Reduction

On the arrival of a new vector pair, we first add all the edges corresponding to the new input (if they do not already exist), and then remove the previous vector pair's edges, as opposed to the usual convention of first deleting the previous edges and inserting the new ones. This ensures that the graph remains an expander at all steps. The proof of constant expansion is involved, and we defer it to the full version. Since number of edges in a constant-degree graph is $O(N)$, we get the following theorem for expanders.

► **Theorem 11.** *For any constant $\epsilon > 0$, there is no dynamic algorithm maintaining a maximum matching or determining the existence of a perfect matching, on all N -node graphs with constant degree and constant expansion, with amortized $O(N^{1/2-\epsilon})$ update time and $O(N^{1-\epsilon})$ query time, unless the OMv conjecture is false.*

Proof. Consider the reduction graph above. It consists of $N = 16n^2 + 16n + 2 = \Theta(n^2)$ nodes. Every time we get a new (u, v) input vector pair, we update $L_2 \times L_3$ and $R_2 \times R_3$ as detailed above. This takes $O(n)$ updates in total. After that, we query once for the size of the maximum matching in this new graph, and return 1 if and only if $|C| = \frac{N}{2}$.

65:12 Fine-Grained Complexity Lower Bounds for Families of Dynamic Graphs

Thus for each pair of input vectors, we perform $O(n)$ updates and $O(1)$ query. In total, checking n vector pairs takes us $O(n^2)$ updates and $O(n)$ query. If there were an algorithm for maximum matching on constant-degree graphs with update time $O(N^{1/2-\epsilon})$ (i.e., $O(n^{1-2\epsilon})$) and query time $O(N^{1-\epsilon})$ (i.e., $O(n^{2-2\epsilon})$), then we can decide if $uMv = 1$ for all n pairs in $O(n^{3-2\epsilon})$ time, contradicting the OMv conjecture. ◀

4 Other Lower Bounds

Dynamic Maximum Matching. All our lower bounds for the dynamic maximum matching problem are summarized in Theorem 12. Note that $m = O(N)$ for the first three graph classes below.

► **Theorem 12.** *For any constant $\epsilon > 0$, there is no dynamic algorithm for maintaining the size of the maximum matching on the following graph classes, with the amortized update time $u(N) = O(N^{1/2-\epsilon})$ and amortized query time $q(N) = O(N^{1-\epsilon})$,*

1. $\Delta \leq 3$,
 2. constant degree, constant expansion,
 3. power-law graph,
 4. $\Delta \leq O(N^t)$ with $u(N) = O(N^{(1+t)/2-\epsilon})$, and $q(N) = O(N^{1+t-\epsilon})$,
- unless the OMv conjecture is false.

Dynamic Densest Subgraph. Our lower bounds for the dynamic densest subgraph problem are summarized in Theorem 13.

► **Theorem 13.** *For any constant $\epsilon > 0$, there is no dynamic algorithm for maintaining the density of the densest subgraph on the following graph classes, with the amortized update time $u(N) = O(N^{1/4-\epsilon})$ and amortized query time $q(N) = O(N^{1/2-\epsilon})$,*

1. $\Delta \leq 7$,
 2. constant degree, constant expansion,
 3. power-law graph,
- unless the OMv conjecture is false.

Dynamic (s, t) -distance. Our lower bounds for the dynamic (s, t) -shortest paths problem are summarized in Theorem 14, and extend naturally to the SSSP and the APSP problems. Note that $m = O(N)$ for the first four classes below.

► **Theorem 14.** *For any constant $\epsilon > 0$, there is no dynamic algorithm for maintaining (s, t) -distance, SSSP, or APSP on the following graph classes, even for bipartite graphs, with the amortized update time $u(N) = O(N^{1/2-\epsilon})$ and amortized query time $q(N) = O(N^{1-\epsilon})$ (except for the last class), unless the OMv conjecture is false.*

1. $\Delta \leq 3$,
2. $(3 - \delta)$ -approximation, $\Delta \leq 3$,
3. constant degree, constant expansion,
4. power law graph,
5. $\Delta \leq O(N^t)$ with $u(N) = O(N^{(1+t)/2-\epsilon})$, and $q(N) = O(N^{1+t-\epsilon})$,

Partially Dynamic Algorithms. We summarize our lower bounds for partially dynamic algorithms in Theorem 15.

► **Theorem 15.** *For any constant $\epsilon > 0$, there is no algorithm in either the insertions-only or deletions-only setting maintaining (s, t) -distance or maximum matching, on all N -node graphs with maximum degree $\Delta \leq 3$, with amortized $O(N^{1/2-\epsilon})$ update time and $O(N^{1-\epsilon})$ query time, unless the OMv conjecture is false.*

References

- 1 Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *FOCS*, pages 477–486. IEEE Computer Society, 2016.
- 2 Ittai Abraham, Shiri Chechik, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. On dynamic approximate shortest paths for planar graphs with worst-case costs. 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 740–753. SIAM, 2016. doi:10.1137/1.9781611974331.ch53.
- 3 Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, STOC '12*, pages 1199–1218, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2213977.2214084.
- 4 William Aiello, Fan Chung Graham, and Linyuan Lu. A random graph model for power law graphs. *Exp. Math.*, 10(1):53–66, 2001. doi:10.1080/10586458.2001.10504428.
- 5 Panagiotis Charalampopoulos and Adam Karczmarz. Single-source shortest paths and strong connectivity in dynamic planar graphs. In *ESA*, volume 173 of *LIPIcs*, pages 31:1–31:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 6 Julia Chuzhoy. Decremental all-pairs shortest paths in deterministic near-linear time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 626–639. ACM, 2021. doi:10.1145/3406325.3451025.
- 7 Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2478–2496. SIAM, 2021. doi:10.1137/1.9781611976465.147.
- 8 Søren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. *CoRR*, abs/1602.06705, 2016. arXiv:1602.06705.
- 9 Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2212–2228. SIAM, 2021. doi:10.1137/1.9781611976465.132.
- 10 Fan Chung Graham. Large dynamic graphs: What can researchers learn from them? *SIAM News*, 37(3), 2004.
- 11 Manoj Gupta and Richard Peng. Fully dynamic $(1 + \epsilon)$ -approximate matchings. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 548–557, 2013. doi:10.1109/FOCS.2013.65.
- 12 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30. ACM, 2015.
- 13 Monika Henzinger, Andrea Lincoln, and Barna Saha. The complexity of average-case dynamic subgraph counting. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–498. SIAM, 2022.
- 14 Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997. doi:10.1006/jcss.1997.1493.

- 15 Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, and Piotr Sankowski. Incremental single-source reachability in planar digraphs. 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 1108–1121. ACM, 2017. doi:10.1145/3055399.3055480.
- 16 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. 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 313–322. ACM, 2011. doi:10.1145/1993636.1993679.
- 17 Wenyu Jin and Xiaorui Sun. Fully dynamic c-edge connectivity in subpolynomial time. *CoRR*, abs/2004.07650, 2020. arXiv:2004.07650.
- 18 P. N. Klein and S. Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, November 1998. doi:10.1007/PL00009223.
- 19 Saurabh Sawlani and Junxing Wang. Near-optimal fully dynamic densest subgraph. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 181–193, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384327.
- 20 Sairam Subramanian. A fully dynamic data structure for reachability in planar digraphs. In Thomas Lengauer, editor, *Algorithms - ESA '93, First Annual European Symposium, Bad Honnef, Germany, September 30 - October 2, 1993, Proceedings*, volume 726 of *Lecture Notes in Computer Science*, pages 372–383. Springer, 1993. doi:10.1007/3-540-57273-2_72.

$O(1)$ Steiner Point Removal in Series-Parallel Graphs

D. Ellis Hershkowitz ✉

Carnegie Mellon University, Pittsburgh, PA, USA

Jason Li ✉

Berkeley, CA, USA

Abstract

We study how to vertex-sparsify a graph while preserving both the graph’s metric and structure. Specifically, we study the Steiner point removal (SPR) problem where we are given a weighted graph $G = (V, E, w)$ and terminal set $V' \subseteq V$ and must compute a weighted minor $G' = (V', E', w')$ of G which approximates G ’s metric on V' . A major open question in the area of metric embeddings is the existence of $O(1)$ multiplicative distortion SPR solutions for every (non-trivial) minor-closed family of graphs. To this end prior work has studied SPR on trees, cactus and outerplanar graphs and showed that in these graphs such a minor exists with $O(1)$ distortion.

We give $O(1)$ distortion SPR solutions for series-parallel graphs, extending the frontier of this line of work. The main engine of our approach is a new metric decomposition for series-parallel graphs which we call a hammock decomposition. Roughly, a hammock decomposition is a forest-like structure that preserves certain critical parts of the metric induced by a series-parallel graph.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners; Theory of computation → Shortest paths; Theory of computation → Approximation algorithms analysis

Keywords and phrases Metric embeddings, graph algorithms, vertex sparsification

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.66

Related Version *Full Version:* <https://arxiv.org/pdf/2104.00750.pdf>

Funding Supported in part by NSF grants CCF-1527110, CCF-1618280, CCF-1814603, CCF-1910588, NSF CAREER award CCF-1750808, a Sloan Research Fellowship, funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (ERC grant agreement 949272) and Swiss National Foundation (project grant 200021-184735).

D. Ellis Hershkowitz: Supported by the Air Force Office of Scientific Research under award number FA9550-20-1-0080.

1 Introduction

Graph sparsification and metric embeddings aim to produce compact representations of graphs that approximately preserve desirable properties of the input graph. For instance, a great deal of work has focused on how, given some input graph G , we can produce a simpler graph G' whose metric is a good proxy for G ’s metric; see, for example, work on tree embeddings [4, 16], distance oracles [36, 35] and graph spanners [2, 1] among many other lines of work. Simple representations of graph metrics enable faster and more space efficient algorithms, especially when the input graph is very large. For this reason these techniques are the foundation of many modern algorithms for massive graphs.



© D. Ellis Hershkowitz and Jason Li;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 66; pp. 66:1–66:17



Leibniz International Proceedings in Informatics

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

Compact representations of graphs are particularly interesting when we assume that G is a member of a minor-closed graph family such as tree, cactus, series-parallel or planar graphs.¹ As many algorithmic problems are significantly easier on such families – see e.g. [26, 3, 37] – it is desirable that G' is not only a simple approximation of G 's metric but that it also belongs to the same family as G .

Steiner point removal (SPR) formalizes the problem of producing a simple G' in the same graph family as G that preserves G 's metric. In SPR we are given a weighted graph $G = (V, E, w)$ and a terminal set $V' \subseteq V$ where $V \setminus V'$ are called “Steiner points.” We must return a weighted graph $G' = (V', E', w')$ where:

1. G' is a minor of G ;
2. $d_G(u, v) \leq d_{G'}(u, v) \leq \alpha \cdot d_G(u, v)$ for every $u, v \in V'$;

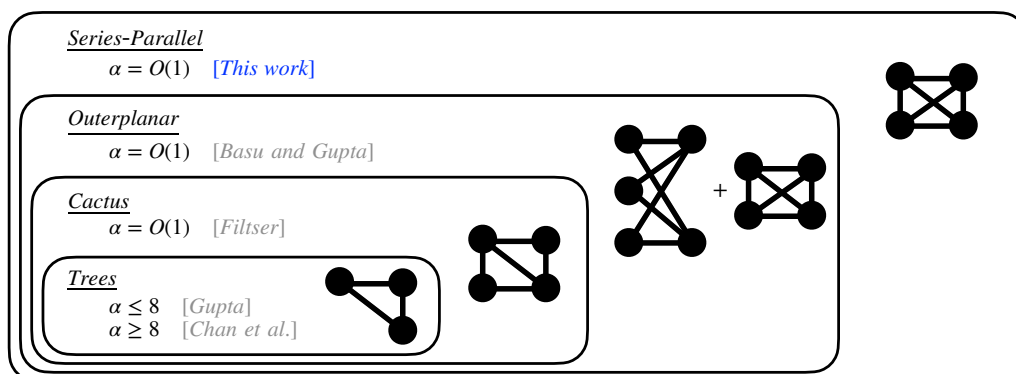
and our aim is to minimize the multiplicative distortion α . We refer to a G' with distortion α as an α -SPR solution. In the above d_G and $d_{G'}$ give the distances in G and G' respectively.

If we only required that G' satisfies the second condition then we could always achieve $\alpha = 1$ by letting G' be the complete graph on V' where $w'(\{u, v\}) = d_G(u, v)$ for every $u, v \in V'$. However, such a G' forfeits any nice structure that G may have exhibited. Thus, the first condition ensures that if G belongs to a minor-closed family then so does G' . The second condition ensures that G' 's metric is a good proxy for G 's metric. G' is simpler than G since it is a graph only on V' while G' is a proxy for G 's metric by approximately preserving distances on V' .

As [22] observed, even for the simple case of trees we must have $\alpha > 1$. For example, consider the star graph with unit weight edges where V' consists of the leaves of the star. Any tree $G' = (V', E', w')$ has at least two vertices u and v whose connecting path consists of at least two edges. On the other hand, the length of any edge in G' is at least 2 and so $d_{G'}(u, v) \geq 4$. Since $d_G(u, v) = 2$ it follows that $\alpha \geq 2$. While this simple example rules out the possibility of 1-SPR solutions on trees, it leaves open the possibility of small distortion solutions for minor-closed families.

In this vein several works have posed the existence of $O(1)$ -SPR solutions for minor-closed families as an open question: see, for example, [5, 19, 8, 30, 11] among other works. A line of work (summarized in Figure 1) has been steadily making progress on this question for the past two decades. [22] showed that trees (i.e. connected K_3 -minor-free graphs) admit 8-SPR solutions.[20] recently gave a simpler proof of this result. [8] proved this was tight by showing that $\alpha \geq 8$ for trees which remains the best known lower bound for K_h -minor-free graphs. In an exciting recent work, [19] reduced $O(1)$ -SPR in K_h -minor-free graphs to computing “ $O(1)$ scattering partitions” and showed how to compute these partitions for several graph classes, including cactus graphs (i.e. all connected F -minor-free graphs where F is K_4 missing one edge). Lastly, a work of [5] generalizes these results by showing that outerplanar graphs (i.e. graphs which are both K_4 and $K_{2,3}$ -minor-free) have $\alpha = O(1)$ solutions.

¹ A graph G' is a minor of a graph G if G' can be attained (up to isomorphism) from G by edge contractions as well as vertex and edge deletions. A graph is F -minor-free if it does not have F as a minor. A family of graphs \mathcal{G} is said to be minor-closed if for any $G \in \mathcal{G}$ if G' is a minor of G then $G' \in \mathcal{G}$. A seminal work of Robertson and Seymour [34] demonstrated that every minor-closed family of graphs is fully characterized by a finite collection of “forbidden” minors. In particular, if \mathcal{G} is a minor-closed family then there exists a finite collection of graphs \mathcal{M} where $G \in \mathcal{G}$ iff G does not have any graph in \mathcal{M} as a minor. Here and throughout this work we will use “minor-closed” to refer to all non-trivial minor-closed families of graphs; in particular, we exclude the family of all graphs which is minor-closed but trivially so.



■ **Figure 1** A summary of the SPR distortion for (connected) K_h -minor-free graphs achieved in prior work and our own. Graph classes illustrated according to containment. We also give the forbidden minors for each graph family.

1.1 Our Contributions

In this work, we advance the state-of-the-art for Steiner point removal in minor-closed graph families. We show that series-parallel graphs (i.e. graphs which are K_4 -minor-free) have $O(1)$ -SPR solutions. The following theorem summarizes the main result of our paper.

► **Theorem 1.** *Every series-parallel graph $G = (V, E, w)$ with terminal set $V' \subseteq V$ has a weighted minor $G' = (V', E', w')$ such that for any $u, v \in V'$ we have*

$$d_G(u, v) \leq d_{G'}(u, v) \leq O(1) \cdot d_G(u, v).$$

Moreover, G' is poly-time computable by a deterministic algorithm.

Series-parallel graphs are a strict superset of all of the aforementioned graph classes for which $O(1)$ -SPR solutions were previously known; again, see Figure 1. Series-parallel graphs are one of the most well-studied graph classes in metric embeddings and serve as a frequent test bed for making progress on long-standing open questions. For example, series-parallel graphs are one of the few graph classes for which the well-known GNRS conjecture in metric embeddings has been successfully proven [25]. For further examples see, among many other works, those of [23, 7] and [13].

Relation to Prior Results

From a metric-embeddings perspective, series-parallel graphs are significantly more complex than outerplanar graphs (the largest minor-free graph class for which $O(1)$ -distortion Steiner point removal was known prior to our work). For example, [24] showed that outerplanar graphs can be embedded into “dominating tree metrics” with constant distortion but that such an embedding for series-parallel graphs incurs $\Omega(\log n)$ distortion. Likewise, outerplanar graphs embed isometrically into l_1 which is known to not be possible for series-parallel graphs; see [33] and [9] for details. Thus, the metrics induced by series-parallel graphs often behave very differently and in less well-structured ways than those induced by outerplanar graphs.

Furthermore, the techniques on which we rely are quite different than those of [5] for the outerplanar case. At least two aspects of these techniques may be of independent interest. We defer a more thorough overview of our techniques to Section 4 but briefly highlight these two points now.

A New Approach for Steiner Point Removal

First, much of our approach generalizes to any K_h -minor-free graph so our approach seems like a promising avenue for future work on $O(1)$ -SPR in minor-closed families. Specifically, we prove our result by beginning with the “chops” used by [29] to build low diameter decompositions for K_h -minor-free graphs. For input $\Delta > 0$ and root vertex r , these chops consist of deleting any edge which for some $i \in \mathbb{Z}$ has endpoints at distance $i\Delta$ and $i\Delta + 1$ from r ; removing such edges partitions the input graph into width Δ “annuli.” We begin with these chops but then slightly perturb them to respect the shortest path structure of the graph, resulting in what we call $O(1)$ -scattering chops. We argue that the result of repeating such scattering chops is a scattering partition which by the results of [19] can be used to construct an $O(1)$ -SPR solution.

The key to this strategy is arguing that series-parallel graphs admit a certain structure – which we call a hammock decomposition – that enables one to perform these perturbations in a principled way. If one could demonstrate a similar structure for K_h -minor-free graphs or otherwise demonstrate the existence of $O(1)$ -scattering chops for such graphs, then the techniques laid out in our work would immediately give $O(1)$ -SPR solutions for all K_h -minor-free graphs.

New Geometric Structure for Series-Parallel Graphs

Second, our hammock decompositions are a new metric decomposition for series-parallel graphs which may be interesting in their own right. We give significantly more detail in the full version of our work but briefly summarize our decomposition for now. We show that for any fixed BFS tree T_{BFS} there is a forest-like subgraph which contains all shortest paths between cross edges of T_{BFS} .² Specifically, the “nodes” of this forest are not vertices but highly structured subgraphs of the input series-parallel graph which we call hammock graphs. A hammock graph consists of two subtrees of the BFS tree and the cross edges between them.

Our hammock decompositions stand in contrast to the fact that the usual way in which one embeds a graph into a tree – by way of dominating tree metrics – are known to incur distortion $\Omega(\log n)$ in series-parallel graphs [25]. Furthermore, our decomposition can be seen as a metric-strengthening of the classic nested ear decompositions for series-parallel graphs of [28] and [15]. In general, a nested ear decomposition need not reflect the input metric. However, not only can one almost immediately recover a nested ear decomposition from a hammock decomposition, but the output nested ear decomposition interacts with the graph’s metric in a highly structured way (see the full version of our work).

Open Questions Resolved

Lastly, we note that, in addition to making progress on the existence of $O(1)$ -SPR solutions for every minor-closed family, our work also settles several open questions. The existence of $O(1)$ -SPR solutions for series-parallel graphs was stated as an open question by both [5] and [8]; our result answers this question in the affirmative. Furthermore, [20] posed the existence of $O(1)$ scattering partitions for outerplanar and series-parallel graphs as an open question; we prove our main result by showing that series-parallel graphs admit $O(1)$ scattering partitions, settling both of these questions.

² Here and throughout this work a cross edge is an edge that is in the input graph but not in T_{BFS} .

2 Related Work

We briefly review additional related work.

Since the introduction of SPR by [22], a variety of works have studied the bounds achievable for well-behaved families of graphs for several very similar problems. [30] studied a problem like SPR but where distances in G must be exactly preserved by G' and the number of Steiner vertices – that is, vertices not in V' – must be made as small as possible; this work showed that while $O(k^4)$ Steiner vertices suffice (where $k = |V'|$) for general graphs, better bounds are possible for well-behaved families of graphs. More generally, [11] studied how to trade off between the number of terminals and distortion of G' , notably showing $(1 + \epsilon)$ distortion is possible in planar graphs with $\tilde{O}(k^2/\epsilon^2)$ Steiner vertices. [14] showed that in minor-closed graphs distances can be preserved up to $O(1)$ multiplicative distortion in expectation by a distribution over minors as opposed to preserving distances deterministically with a single minor as in SPR.

A variety of recent works have also studied how to find minors which preserve properties of G other than G 's metric. [14] studied a flow/cut version of SPR where the goal is for G' to be a minor of G just on the specified terminals while preserving the congestion of multicommodity flows between terminals: this work showed that a convex combination of planar graphs can preserve congestion on V' up to a constant while for general graphs a convex combination of trees preserves congestion up to an $O(\log k)$. Similarly, [31] studied how to find minimum-size planar graphs which preserve terminal cuts. [21] studied how to find a minor of a directed graph with as few Steiner vertices and which preserves the reachability relationships between all k terminals, showing that $O(k^3)$ vertices suffices for general graphs but $O(\log k \cdot k^2)$ vertices suffices for planar graphs.

There has been considerable effort in the past few years on developing good SPR solutions for general graphs. [27] gave $O(\log^5 k)$ -SPR solutions for general graphs. This was improved by [10] who gave $O(\log^2 k)$ -SPR solutions which was, in turn, improved by [18] and [17] who gave $O(\log k)$ -SPR solutions for general graphs. We also note that [19] also achieved similar results by way of scattering partitions, albeit with a worse poly-log factor as well as the first $O(1)$ -SPR solutions for bounded pathwidth graphs.

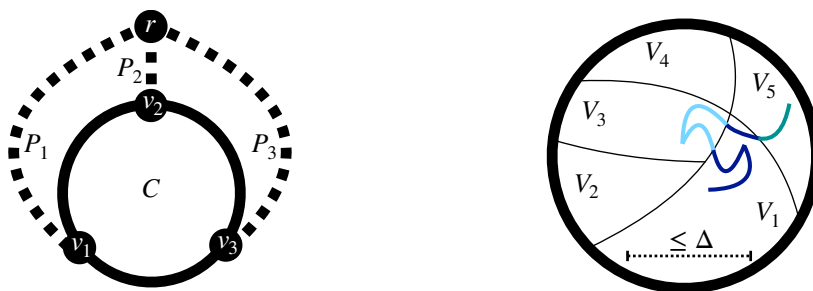
3 Preliminaries

Before giving an overview of our approach we summarize the characterization of series-parallel graphs we use throughout this work as well as the scattering partition framework of [19] on which we build.

3.1 Characterizations of Series-Parallel Graphs

There are some minor inconsistencies in the literature regarding what is considered a series-parallel graph and so we clarify which notion of series-parallel we use throughout this paper. Some works – e.g. [15] – take series parallel graphs to be those which can be computed by iterating parallel and series compositions of graphs. Call these series-parallel A graphs.³

³ The following is a definition of series-parallel A graphs due to [15]. A graph is two-terminal if it has a distinct source s and sink t . Let G and H be two two-terminal graphs with sources s and s' and sinks t and t' . Then the series composition of G and H is the graph resulting from identifying t and s' as the same vertex. The parallel composition of G and H is the graph resulting from identifying s and s' as the same vertex and t and t' as the same vertex. A two-terminal series-parallel graph is a two-terminal graph which is either a single edge or the graph resulting from the series or parallel composition of two two-terminal series-parallel graphs. A graph is series-parallel A if it has some pair of vertices with respect to which it is two-terminal series-parallel.



(a) A clawed cycle. (b) A scattering partition.

■ **Figure 2** In (a) we illustrate a clawed cycle where the cycle C is given in solid black and each path is given in dotted black. In (b) we illustrate a scattering partition with $\tau = 3$ and how one path P of length at most Δ is incident to at most three parts where we color the subpaths of P according to the incident part.

Strictly speaking, series-parallel A graphs are not even minor-closed as they are not closed under edge or vertex deletion. Other works – e.g. [19] – take series-parallel graphs to be graphs whose biconnected⁴ components are each series-parallel A graphs; call these series-parallel B graphs. Series-parallel B graphs clearly contain series-parallel A graphs and, moreover, are minor-closed. For the rest of this work we will use the more expansive series-parallel B notion; henceforth we use “series-parallel” to mean series-parallel B.

It is well-known that a graph is K_4 -minor-free iff it is series-parallel [6]. Similarly a graph has treewidth at most 2 iff it is series-parallel [6]. In this work we will use an alternate definition in terms of “clawed cycles” which we illustrate in Figure 2a.⁵

► **Definition 2** (Clawed Cycle). *A clawed cycle is a graph consisting of a root r , a cycle C and three paths P_1 , P_2 and P_3 from r to vertices $v_1, v_2, v_3 \in C$ where $v_1 \neq v_2 \neq v_3$*

The fact that series-parallel graphs are exactly those that do not have any clawed cycles as a subgraph was proven by [12]; we give a proof for completeness.

► **Lemma 3** ([12]). *A graph G is series-parallel iff it does not contain a clawed cycle as a subgraph.*

Proof. K_4 is itself a clawed cycle and so a graph with no clawed cycle subgraphs is K_4 -minor-free and therefore series-parallel. If a graph contains a clawed cycle then we can construct a K_4 minor by arbitrarily contracting the graph into v_1, v_2, v_3 and r , as defined in Definition 2. ◀

3.2 Scattering Partitions

Our result will be based on a new graph partition introduced by [19], the scattering partition. Roughly speaking, a scattering partition of a graph is a low-diameter partition which respects the shortest path structure of the graph; see Figure 2b.⁶

⁴ A connected component C is biconnected if C remains connected even after the deletion of any one vertex in C .

⁵ We note that clawed cycles are also called “embedded Wheatstone bridge.”

⁶ We drop one of the parameters of the definition of [19] as it will not be necessary for our purposes.

► **Definition 4** (Scattering Partition). *Given a weighted graph $G = (V, E, w)$, a partition $\mathcal{P} = \{V_i\}_i$ of V is a (τ, Δ) scattering partition if:*

1. **Connected:** *Each $V_i \in \mathcal{P}$ is connected;*
2. **Low Weak Diameter:** *For each $V_i \in \mathcal{P}$ and $u, v \in V_i$ we have $d_G(u, v) \leq \Delta$;*
3. **Scattering:** *Every shortest path P in G of length at most Δ satisfies $|\{V_i : V_i \cap P \neq \emptyset\}| \leq \tau$.*

[19] extended these partitions to the notion of a scatterable graph.

► **Definition 5** (Scatterable Graph). *A weighted graph $G = (V, E, w)$ is τ -scatterable if it has a (τ, Δ) -scattering partition for every $\Delta \geq 0$.*

We will say that G is deterministic poly-time τ -scatterable if for every $\Delta \geq 0$ a (τ, Δ) -scattering partition is computable in deterministic poly-time.

As a concrete example of a τ -scatterable graph and as observed by [19] notice that all trees are $O(1)$ -scatterable. In particular, suppose we are given a tree and a $\Delta > 0$. If we fix a root vertex r and then delete any edge which for some $i \in \mathbb{Z}$ has endpoints at distance $\frac{i\Delta}{2}$ and $\frac{i\Delta}{2} + 1$ from r this breaks the input tree into connected components. Each component has diameter at most Δ by construction. Furthermore, it is easy to see that any path of length at most Δ is incident to a constant number of these components and so these components indeed form a scattering partition with $\tau = O(1)$. This construction is essentially a single chop of the aforementioned KPR strategy. However, while a KPR chop can be used to construct scattering partitions on trees, as we will see shortly, KPR chops on series-parallel graphs do not, in general, result in scattering partitions.

Lastly, the main result of [19] is that solving SPR reduces to showing that every induced subgraph is scatterable. In the following $G[A]$ is the subgraph of G induced by the vertex set A .

► **Theorem 6** ([19]). *A weighted graph $G = (V, E, w)$ with terminal set $V' \subseteq V$ has an $O(\tau^3)$ -SPR solution if $G[A]$ is τ -scatterable for every $A \subseteq V$. Furthermore, if $G[A]$ is deterministic poly-time scatterable for every $A \subseteq V$ then the $O(\tau^3)$ -SPR solution is computable in deterministic poly-time.*

4 Intuition and Overview of Techniques

We now give intuition and a high-level overview of our techniques. As discussed in the previous section, solving SPR with $O(1)$ distortion for any fixed graph reduces to showing that the subgraph induced by every subset of vertices is $O(1)$ -scatterable. Moreover, since every subgraph of a K_h -minor-free graph is itself a K_h -minor-free graph, it follows that in order to solve SPR on any fixed K_h -minor-free graph, it suffices to argue that every K_h -minor-free graph is $O(1)$ -scatterable.

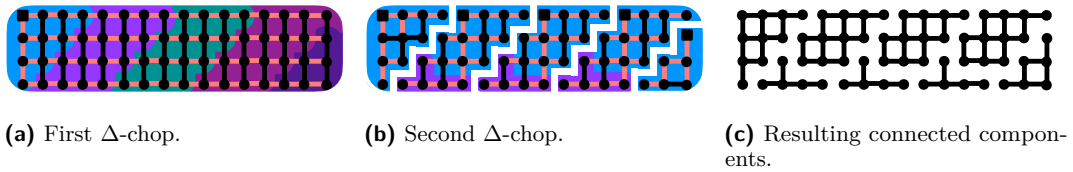
Thus, the fact that we dedicate the rest of this document to showing is as follows.

► **Theorem 7.** *Every series-parallel graph G is deterministic, poly-time $O(1)$ -scatterable.*

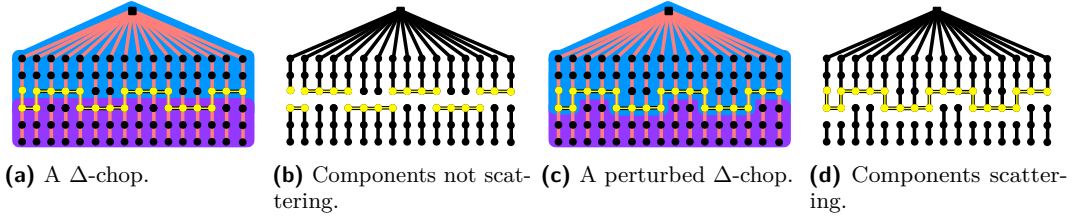
Combining this with Theorem 6 immediately implies Theorem 1.

4.1 General Approach

Given a series-parallel graph G and some $\Delta \geq 1$, our goal is to compute an $(O(1), \Delta)$ -scattering partition for G . Such a partition has two non-trivial properties to satisfy: (1) each constituent part must have weak diameter at most Δ and (2) each shortest path of length at most Δ must be in at most $O(1)$ parts (a property we will call “scattering”).



■ **Figure 3** Two levels of Δ -chops on the grid graph for $\Delta = 3$. We give the edges of the BFS trees we use in pink; roots of these trees are given as squares. Background colors give the annuli of nodes.



■ **Figure 4** An example (of an outerplanar graph) where a Δ -chop does not produce a scattering partition but how perturbing said chop does. Here, we imagine that the root is at the top of the graph and each edge incident to the root has length $\Delta - 3$. We highlight the path P that either ends up in many or one connected component depending on whether we perturb our Δ -chop in yellow.

A well-known technique of [29] – henceforth “KPR” – has proven useful in finding so-called low diameter decompositions for K_h -minor-free graphs and so one might reasonably expect these techniques to prove useful for finding scattering partitions. Specifically, KPR shows that computing low diameter decompositions in a K_h -minor-free graph can be accomplished by $O(h)$ levels of recursive “ Δ -chops”. Fix a root r and a BFS tree T_{BFS} rooted at r . Then, a Δ -chop consists of the deletion of every edge with one vertex at depth $i \cdot \Delta$ and another vertex at depth $i \cdot \Delta + 1$ for every $i \in \mathbb{Z}_{\geq 1}$; that is, it consists of cutting edges between each pair of adjacent Δ -width annuli. KPR proved that if one performs a Δ -chop and then recurses on each of the resulting connected component then after $O(h)$ levels of recursive depth in a K_h -minor free graph the resulting components all have diameter at most $O(\Delta)$. We illustrate KPR on the grid graph in Figure 4.

Thus, we could simply apply Δ -chops $O(h)$ times to satisfy our diameter constraints (up to constants) and hope that the resulting partition is also scattering. Unfortunately, it is quite easy to see that (even after just one Δ -chop!) a path of length at most Δ can end up in arbitrarily many parts of the resulting partition. For example, the highlighted shortest path in Figures 4a and 4b repeatedly moves back and forth between two annuli and ends up in arbitrarily many parts after a single Δ -chop. Nonetheless, this example is suggestive of the basic approach of our work. In particular, if we merely perturbed our first Δ -chop to cut “around” said path as in Figures 4c and 4d then we could ensure that this path ends up in a small number of partitions.

More generally, the approach we take in this work is to start with the KPR chops but then slightly perturb these chops so that they do not cut *any* shortest path of length at most Δ more than $O(1)$ times. That is, all but $O(1)$ edges of any such path will have both vertices in the same (perturbed) annulus. We then repeat this recursively on each of the resulting connected components to a constant recursion depth. Since each subpath of a shortest path of length at most Δ is itself a shortest path with length at most Δ , we know that each such shortest path is broken into a constantly-many-more shortest paths at each level of recursion. Moreover, since we recurse a constant number of times, each path ends up in a constant number of components.

Implementing this strategy requires meeting two challenges. First, it is not clear that the components resulting from KPR still have low diameter if we allow ourselves to perturb our chops. Second, it is not clear how to perturb a chop so that it works *simultaneously* for every shortest path. Solving the first challenge will be somewhat straightforward while solving the second will be significantly more involved. In particular, what makes the second challenge difficult is that we cannot, in general, perturb a chop on the basis of one violated shortest path as in the previous example; doing so might cause other paths to be cut too many times which will then require additional, possibly conflicting, perturbations and so on. Rather, we must somehow perturb our chops in a way that takes every shortest path into account all at once.

4.2 Scattering Chops

The easier issue to solve will be how to ensure that our components have low diameter even if we perturb our chops. Here, by closely tracking various constants through a known analysis of KPR we show that the components resulting from KPR with (boundedly) perturbed cuts are still low diameter.

We summarize this fact and the above discussion with the idea of a scattering chop. A (τ, Δ) -scattering chop consists of cutting all edges at *about* every Δ levels in the BFS tree in such a way that no shortest path of length at most Δ is cut more than τ times. Our analysis shows that if all K_h -minor-free graphs admit $(O(1), \Delta)$ -scattering chops for every Δ then they are also $O(1)$ -scatterable and therefore also admit $O(1)$ -SPR solutions; this holds even for $h > 4$.

4.3 Hammock Decompositions and How to Use Them

The more challenging issue we must overcome is how to perturb our chops so that every shortest path of length at most Δ is only cut $O(1)$ times. Moreover, we must do so in a way that does not perturb our boundaries by too much so as to meet the requirements of a scattering chop. We solve this issue with our new metric decomposition for series-parallel graphs, the hammock decomposition.

Consider a shortest path P of length at most Δ . Such a path can be partitioned into a (possibly empty) prefix consisting of only edges in T_{BFS} , a middle portion whose first and last edges are cross edges of T_{BFS} and a (possibly empty) suffix which also only has edges in T_{BFS} . Thus, if we want to compute a scattering chop, it suffices to guarantee that any shortest path of length at most Δ which is either fully contained in T_{BFS} or which is between two cross edges of T_{BFS} is only cut $O(1)$ times by our chop; call the former a BFS path and the latter a cross edge path.

Next, notice that all BFS paths are only cut $O(1)$ times by our initial KPR chops. Specifically, each BFS path can be partitioned into a subpath which goes “up” in the BFS tree and a subpath which goes “down” in the BFS tree. As our initial KPR chops are Δ apart and each such subpath is of length at most Δ , each such subpath is cut at most $O(1)$ times. Thus provided our perturbations do not interfere *too much* with the initial structure of our KPR chops we should expect that our BFS paths will only be cut $O(1)$ times.

Thus, our goal will be to perturb our KPR chops to not cut any cross edge path more than $O(1)$ times while mostly preserving the initial structure of our KPR chops. Our hammock decompositions will allow us to do exactly this. They will have two key components.

The first part is a “forest of hammocks.” Suppose for a moment that our input graph had a forest subgraph F that contained all cross edge paths of our graph which were also shortest paths. Then, it is not too hard to see how to use F to perturb our chops to be scattering for

all cross edge paths. Specifically, for each tree T in our forest F we fix an arbitrary root and then process edges in a BFS order. Edges which we process will be marked or unmarked where initially all edges are unmarked. To process an edge $e = \{u, v\}$ we do the following. If e is marked or u and v both belong to the same annulus then we do nothing. Otherwise, e is unmarked and u is in some annulus A but v is in some other annulus A' (before any perturbation). We then propagate A an additional $\Theta(\Delta)$ deeper into T ; that is if we imagine that v is the child of u in F then we move all descendants of u in F within $\Theta(\Delta)$ of u into A . We then mark all edges in T whose endpoints are descendants of u and within $\Theta(\Delta)$ of u . A simple amortized analysis shows that after performing these perturbations every cross edge path is cut $O(1)$ times: if we think of following a cross edge path from one endpoint to the other, then each time this path is cut there must be at least $\Omega(\Delta)$ many edges we get to traverse until the next time it is cut again.

Unfortunately, it is relatively easy to see that such an F may not exist in a series-parallel graph. The forest of hammocks component of our decompositions is a subgraph which will be “close enough” to such an F , thereby allowing us to perturb our chops similarly to the above strategy. As mentioned in the introduction, a hammock graph consists of two subtrees of a BFS tree and the cross edges between them. A forest of hammocks is a graph partitioned into hammocks where every cycle is fully contained in one of the constituent hammocks. While the above perturbation will guarantee that our cross edge paths are not cut too often, it is not clear that such a perturbation does not change the structure of our initial chops in a way that causes our BFS paths to be cut too many times.

The second part of our hammock decompositions is what we use to guarantee that our BFS paths are not cut too many times by preserving the structure of our initial KPR chops. Specifically, the forest structure of our hammocks will reflect the structure of T_{BFS} . In particular, we can naturally associate each hammock H_i with a single vertex, namely the LCA of any u and v where u is in one tree of H_i and v is in the other. Then, our forest of hammocks will satisfy the property that if hammock H_i is a “parent” of hammock H_j in our forest of hammocks then the LCA corresponding to H_i is an ancestor of the LCA corresponding to H_j in T_{BFS} ; even stronger, the LCA of H_j will be contained in H_i . Roughly, the fact that our forest of hammocks mimics the structure of T_{BFS} in this way will allow us to argue that the above perturbation does not alter the initial structure of our KPR chops too much, thereby ensuring that BFS paths are not cut too many times.

The computation of our hammock decompositions constitutes the bulk of our technical work but is somewhat involved. The basic idea is as follows. We will partition all cross edges into equivalence classes where each cross edge in an equivalence class shares an LCA in T_{BFS} (though there may be multiple, distinct equivalence classes with the same LCA). Each such equivalence class will eventually correspond to one hammock in our forest of hammocks. To compute our forest of hammocks we first connect up all cross edges in the same equivalence class. Next we connect our equivalence classes to one another by cross edge paths which run between them. We then extend our hammocks along paths towards their LCAs to ensure the above-mentioned LCA properties. Finally, we add so far unassigned subtrees of T_{BFS} to our hammocks. We will argue that when this process fails it shows the existence of a K_4 -minor and, in particular, a clawed cycle.

5 Notation and Conventions

Before proceeding to our formal results we specify the notation we use throughout this work as well as some of the assumptions we make on our input series-parallel graph without loss of generality (WLOG).

Graphs. For a weighted graph $G = (V, E, w)$, we let $V(G) = V$ and $E(G) = E$ give the vertex set and edge sets of G respectively. We will sometimes abuse notation and use G to stand for V or E when it is clear from context if we mean G 's vertex or edge set. Our weight function on edges is $w : E \rightarrow \mathbb{Z}_{\geq 1}$. Given graphs G and H , we will use the notation $H \subseteq G$ to indicate that H is a subgraph of G . The weak diameter of a subgraph H is $\max_{u,v \in V(H)} d_G(u, v)$.

Assumption of Unique Shortest Paths and Unit Weights. We will assume throughout this work that in our input series-parallel graph for any vertices u and v the shortest path between u and v is unique and that $w(e) = 1$ for every e . It is easy to see that our algorithms extend to non-unique shortest paths and the non-unit weight edge cases by standard techniques. In particular, one can randomly perturb the initial weights of the input graph so as to guarantee the uniqueness of shortest paths. Similarly, one can expand each edge of weight $w(e)$ into a path of $w(e)$ edges while preserving series-parallelness and the metric on the nodes from the original graph which suffices for our purposes.

Induced Graphs and Edges. Given an edge set E and disjoint vertex sets V_1 and V_2 , we let $E(V_1, V_2) := \{e = \{v_1, v_2\} \in E : v_1 \in V_1, v_2 \in V_2\}$ be all edges between V_1 and V_2 . Given graph $G = (V, E)$ and a vertex set $U \subseteq V$, we let $G[U] = (U, E_U)$ be the ‘‘induced subgraph’’ of G where $\{u', v'\} \in E_U$ iff $\{u', v'\} \in E$. Given a collection of subgraphs $\mathcal{H} = \{H_i\}_i$ of a graph we call $G[\mathcal{H}] := (\bigcup_i V(H_i), \bigcup_i E(H_i))$ the induced subgraph of \mathcal{H} . Similarly, we will let $E(\mathcal{H}) := \bigcup_i E(H_i)$ give the edges of \mathcal{H} . We emphasize that it is not necessarily the case that $G[\mathcal{H}] = G[V(\mathcal{H})]$.

Paths. Given a path $P = (v_0, v_1, \dots, v_k, v_{k+1})$ we will use $\text{internal}(P) := \{v_1, \dots, v_k\}$ to refer to the internal vertices of P . We will say that a path P is between two vertex sets U and W if its first and last vertices are in U and W respectively and $\text{internal}(P) \cap U = \emptyset$ and $\text{internal}(P) \cap W = \emptyset$. We will sometimes abuse notation and use P and $E(P)$ interchangeably. We will also sometimes say such a path is ‘‘from’’ U to W interchangeably with a path is ‘‘between’’ U and W . We will use $P \oplus P'$ to refer to the concatenation of two paths which share an endpoint throughout this paper. For a tree T , we will let $T(u, v)$ stand for the unique path between u and v in T for $u, v \in V(T)$. We will sometimes assume that a path from a vertex set to another vertex set is directed in the natural way.

BFS Tree Notation. For much of this work we will fix a series-parallel graph $G = (V, E)$ along with a fixed but arbitrary root $r \in V$ and a fixed but arbitrary BFS tree T_{BFS} with respect to r . When we do so we will let $E_c := E \setminus E(T_{\text{BFS}})$ be all cross edges of T_{BFS} . For $u, v \in V$, if $u \in T_{\text{BFS}}(r, v) \setminus \{v\}$ then we say that u is an ancestor of v . In this case, we also say that v is a descendant of u . If u is an ancestor of v or v is an ancestor of u then we say that u and v are related; otherwise, we say that u and v are unrelated. For two vertices $u, v \in V$ we will use the notation $u \prec v$ to indicate that v is an ancestor of u and we will use the notation $u \preceq v$ to indicate that v is an ancestor of or equal to u . It is easy to verify that \preceq induces a partial order. We let $T_{\text{BFS}}(v) := T_{\text{BFS}}[\{v\} \cup \{u \in V : u \text{ is a descendant of } v\}]$ be the subtree of T_{BFS} rooted at v . Given a connected subgraph $T \subseteq T_{\text{BFS}}$, we will let $\text{high}(T)$ be the vertex in $V(T)$ which is an ancestor of all vertices in $V(T)$. Given a path $P \subseteq T_{\text{BFS}}$ we will say that P is monotone if $\text{high}(P)$ is an ancestor of all vertices in P and there is some vertex $\text{low}(P)$ which is a descendant of all vertices in P . We let $h(v)$ give the height of a vertex in T_{BFS} (where we imagine that the nodes furthest from r are at height 0). We let $\text{LCA}(e)$ be the least common ancestor of u and v in T_{BFS} for each $e = \{u, v\} \in E$.

Miscellaneous. We will use \sqcup for disjoint set union throughout this paper. That is $A \sqcup B$ is equal to $A \cup B$ but indicates that $A \cap B = \emptyset$.

6 Perturbing KPR and Scattering Chops

In this section we show that KPR still gives low diameter components even if its boundaries are perturbed and therefore somewhat “fuzzy.” We then observe that this fact shows that “ $O(1)$ -scattering chops” imply the existence of $O(1)$ -scattering partitions for K_h -minor-free graphs and therefore $O(1)$ -SPR solutions.

6.1 Perturbing KPR

We will repeatedly take the connected components of annuli with “fuzzy” boundaries. We formalize this with the idea of a c -fuzzy Δ -chop; see Figure 5a for an illustration.

► **Definition 8** (*c-Fuzzy Δ -Chop*). *Let $G = (V, E, w)$ be a weighted graph with root r and fix $0 \leq c < 1$ and $\Delta \geq 1$. Then a c -fuzzy Δ -chop is a partition of V into “fuzzy annuli” $\mathcal{A} = \{A_1, A_2, \dots\}$ where for every i and $v \in A_i$ we have*

$$(i-1)\Delta - \frac{c\Delta}{2} \leq d(r, v) < i \cdot \Delta + \frac{c\Delta}{2}.$$

As each fuzzy annulus in a fuzzy chop may contain many connected components we must be careful when specifying how recursive application of these chops break a graph into connected components; hence the following definitions. Given fuzzy annulus A_i , we let \mathcal{C}_i be the connected components of A_i .

► **Definition 9** (*Components Resulting from a c -Fuzzy Δ -Chop*). *Let $G = (V, E, w)$ be a weighted graph and let \mathcal{C} be a partition of V into connected components. Then we say that \mathcal{C} results from one level of c -fuzzy Δ -chops if there is a c -fuzzy Δ -chop \mathcal{A} with respect to some root $r \in V$ satisfying $\mathcal{C} = \bigcup_{i: A_i \in \mathcal{A}} \mathcal{C}_i$. Similarly, for $h \geq 2$ we say that \mathcal{C} results from h -levels of c -fuzzy Δ -chops if there is some \mathcal{C}' which results from one level of c -fuzzy Δ -chops and \mathcal{C} is the union of the result of $h-1$ levels of c -fuzzy Δ -chops on each $C' \in \mathcal{C}'$.*

We will now claim that taking $h-1$ levels of fuzzy chops in a K_h -minor-free graph will result in a connected, low weak diameter partition. In particular, we show the following lemma, the main result of this section.

► **Lemma 10.** *Let Δ and h satisfy $2 \leq h$, $\Delta \geq 1$ and fix constant $0 \leq c < 1$. Suppose \mathcal{C} is the result of $h-1$ levels of c -fuzzy Δ -chops in a K_h -minor-free weighted graph G . Then, the weak diameter of every $C \in \mathcal{C}$ is at most $O(h \cdot \Delta)$.*

For the rest of this section we identify the nodes of a minor of graph G with “supernodes.” In particular, we will think of each of the vertices of the minor as corresponding to a disjoint, connected subset of vertices in G (a supernode) where the minor can be formed from G (up to isomorphism) by contracting the constituent nodes of each such supernodes.

Our proof will closely track a known analysis of KPR [32]. The sketch of this strategy is as follows. We will argue that if we fail to produce parts with low diameter then we have found K_h as a minor. Our proof will be by induction on the number of levels of fuzzy chops. Suppose \mathcal{C} is produced by $h-1$ levels of fuzzy chops; in particular, suppose \mathcal{C} is produced by taking some fuzzy chop to get \mathcal{C}' and then taking $h-2$ levels of fuzzy chops on each $C' \in \mathcal{C}'$. Also assume that there is some $C \in \mathcal{C}$ which has *large* diameter. Then, C must result from taking $h-2$ levels of fuzzy chops on some $C' \in \mathcal{C}'$ where C' lies in some fuzzy

annulus A_i of G . By our inductive hypothesis it follows that C contains K_{h-1} as a minor. Our goal is to add one more supernode to this minor to get a K_h minor. We will do so by finding disjoint paths of length $O(\Delta)$ in the annulus above A_i from each of the K_{h-1} supernodes all of which converge on a single connected component. By adding these paths to their respective supernodes in the K_{h-1} minor and adding the connected component on which these paths converge to our collection of supernodes, we will end up with a K_h minor.

The main challenge in this strategy is to show how to find paths as above which are disjoint. We will do so by choosing these paths from a “representative” from each supernode where initially the representatives are $\Omega(h\Delta)$ far-apart and grow at most $O(\Delta)$ closer at each level of chops; since we do at most $O(h)$ levels of chops, the paths we choose will never intersect.

To formalize this strategy we must state a few definitions which will aid in arguing that these representatives are far apart.

► **Definition 11** (Δ -Dense). *Given sets $S, U \subseteq V$ we say S is Δ -dense in U if $d(u, S) \leq \Delta$ for every $u \in U$.*

► **Definition 12** (R -Represented). *A K_h minor is R -represented by set $S \subseteq V$ if each supernode $V_i \subseteq V$ of the minor in G contains a representative $v_i \in S \cap V_i$ and these representatives are pairwise at least R apart in G .*

Since V is clearly $(1+c)\Delta$ -dense in V , we can set S to V and j to $h-1$ in the following lemma to get Lemma 10.

► **Lemma 13.** *Fix $0 \leq c < 1$ and $h > j \geq 0$. Let S be any set which is $(1+c)\Delta$ -dense in V and suppose \mathcal{C} is the result of j levels of c -fuzzy Δ -chops and some $C \in \mathcal{C}$ has weak diameter more than $22h\Delta$. Then there exists a K_{j+1} minor which is $8(h-j)\Delta$ -represented by S .*

Proof. Our proof is by induction on j . The base case of $j = 0$ is trivial as K_1 is a minor of any graph with a supernode $+\infty$ -represented by any single vertex in V .

Now consider the inductive step on graph $G = (V, E)$. Fix some set S which is $(1+c)\Delta$ -dense in V and let \mathcal{C} be the result of j levels of c -fuzzy chops using root r with some $C' \in \mathcal{C}$ of diameter more than $22h\Delta$. Suppose C' is in fuzzy annulus A_k and suppose that C' is the result of applying $j-1$ levels of c -fuzzy chops to some C which resulted from 1 level of c -fuzzy chops in G ; note that C is a connected component of A_k and that C' is contained in C .

As an inductive hypothesis we suppose that any $j-1$ levels of c -fuzzy Δ -chops on any graph H which results in a cluster of weak diameter more than $22h\Delta$ demonstrates the existence of a K_j minor in H which is $8(h-j+1)\Delta$ -represented by any set S' which is $(1+c)\Delta$ -dense in $V(H)$. Here weak diameter is with respect to the distances induced by the original input graph.

Thus, by our inductive hypothesis we therefore know that C contains a K_j minor which is $8(h-j+1)\Delta$ -represented by any $S' \subseteq V(C)$ which is $(1+c)\Delta$ -dense in $V(C)$. In particular, we may let S' be the “upper boundary” of C ; that is, we let S' be all vertices v in C such that the shortest path from v to r does not contain any vertices in C . Clearly the shortest path from any vertex in C to r intersects a node in S' ; moreover, when restricted to C this shortest path has length at most $\Delta + c\Delta$ (since C is contained in A_k) which is to say that S' is $(1+c)\Delta$ -dense in C . Thus, by our induction hypothesis there is a K_j minor in C which is $8(h-j+1)\Delta$ -represented by S' . Let V_1, \dots, V_j be the nodes in the supernodes of the K_j minor.

66:14 $O(1)$ Steiner Point Removal in Series-Parallel Graphs

We now describe how to extend the K_j minor to a K_{j+1} minor which is $8(h-j)\Delta$ -represented by S . We may assume that $k \geq 9h+1$; otherwise the distance from every node in A_k to r would be at most $(9h+1)\Delta + \frac{c\Delta}{2} \leq (9h + \frac{3}{2})\Delta$ and so the weak diameter of C' would be at most $(18h+3)\Delta \leq 21h\Delta$, contradicting our assumption on C' 's diameter. It follows that for every $v \in A_k$ we have

$$d(v, r) \geq (k-1)\Delta - \frac{c\Delta}{2} \geq 9h\Delta - \frac{c\Delta}{2} \geq 8h\Delta. \quad (1)$$

We first describe how we grow each supernode V_i from the K_j minor to a new supernode V'_i . Let v_i be the representative in S' for V_i . Consider the path which consists of following the shortest path from v_i to r for distance 2Δ and then continuing on to the nearest node in S ; let v'_i be this nearest node; this path from v_i to v'_i has length at most $(3+c)\Delta$ since S is $(1+c)\Delta$ -dense in $V(G)$. Let V'_i be the union of V_i with the vertices in this path. Since each of these paths is of length at most $(3+c)\Delta \leq 4\Delta$, it follows that each of these paths for each i must be disjoint since each v_i is at least $8(h-j+1)\Delta > 8\Delta$ apart. Further, every v'_i must also, therefore, be at least $8(h-j)\Delta$ apart. Therefore, we let these v'_i form the representatives in S for each of the V'_i .

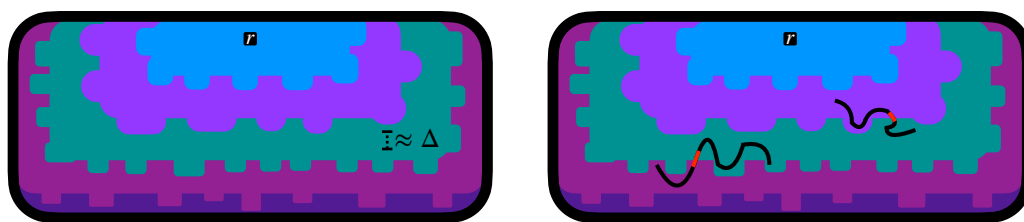
We now describe how we construct the additional supernode, V_0 , which we add to our minor to get a K_{j+1} minor. V_0 will “grow” from the root to S and each of the V'_i . In particular, let $u_i \in V'_i$ be the node in V'_i which is closest to r and let P_i be the shortest path from r to u_i , excluding u_i . Similarly, let v'_0 be the node in S closest to r and let P_0 be the shortest path from r to v'_0 , including v'_0 . Then, we let V_0 be $P_0 \cup P_1 \cup \dots \cup P_j$ and we let v'_0 be the representative for V_0 in S . We claim that for every $i \geq 1$ we have

$$d(P_0, V'_i) \geq 8(h-j)\Delta. \quad (2)$$

In particular, notice that since S is $(1+c)\Delta$ -dense in $V(G)$ we know that $d(r, v'_0) \leq (1+c)\Delta \leq 2\Delta$ and since $d(v, r) \geq 8h\Delta$ for every $v \in A_k$ by Equation (1) and $d(v'_i, A_k) \leq (3+c)\Delta \leq 4\Delta$, it follows that $d(P_0, V'_i) \geq (8h-6)\Delta \geq 8(h-j)\Delta$. Consequently, $d(v'_0, v'_i) \geq 8(h-j)\Delta$ for every $i \geq 1$. Thus, our representatives of each supernode are appropriately far apart.

It remains to show that our supernodes indeed form a K_{j+1} minor; clearly by construction they are all pair-wise adjacent and so it remains only to show that they are all disjoint from one another. We already argued above that for $i, j \geq 1$ any V'_i and V'_j are disjoint so we need only argue that V'_0 is disjoint from each V'_i for $i \geq 1$. P_0 must be disjoint from each V'_i for $i \geq 1$ by Equation (2) and so we need only verify that P_i is disjoint from V'_j for $i, j \geq 1$;

By construction if $i = j$ we know that P_i is disjoint from V'_j so we assume $i \neq j$ and that P_i intersects V'_j for the sake of contradiction. Notice that each P_i has length at most $k\Delta + \frac{c\Delta}{2} - 2\Delta = k\Delta + c\Delta - 2\Delta - \frac{c\Delta}{2} < (k-1)\Delta - \frac{c\Delta}{2}$ by how we construct V'_i . Thus, P_i must be disjoint from A_k . It follows that if P_i intersects V'_j then it must intersect $V'_j \setminus V_j$. However, since $d(v_i, v_j) \geq 8(h-j+1)\Delta \geq 16\Delta$ and the length of paths $V'_i \setminus V_i$ and $V'_j \setminus V_j$ are at most 4Δ we know that $d(V'_i \setminus V_i, V'_j \setminus V_j) \geq 8(h-j)\Delta \geq 8\Delta$. Thus, after intersecting $V'_j \setminus V_j$ and then continuing on to a vertex adjacent to $V'_i \setminus V_i$, we know P_i must travel at least 8Δ ; since the vertices of P_i are monotonically further and further from r , and the vertex in $V'_j \setminus V_j$ that P_i intersects must be distance at least $(k-1)\Delta - \frac{c\Delta}{2} - 4\Delta \geq (k-5)\Delta$ from r , then the last vertex of P_i must be distance at least $(k+3)\Delta$ from r , meaning P_i must intersect annulus A_k , a contradiction. \blacktriangleleft

(a) A c -fuzzy Δ -chop.

(b) Visualizing some paths.

■ **Figure 5** A c -fuzzy Δ -chop that is 1-scattering. We draw each fuzzy annulus in a distinct color. In (b) we visualize some shortest paths of length at most Δ and highlight cut edges in red.

6.2 Scattering Chops

Using Lemma 10 we can reduce computing a good scattering partition and therefore computing a good SPR solution to finding what we call a scattering chop. The following definitions are somewhat analogous to Definition 4 and Definition 5. However, notice that the second definition is for a family of graphs (as opposed to a single graph as in Definition 5). We illustrate a τ -scattering chop in Figure 5.

► **Definition 14** (τ -Scattering Chop). *Given a weighted graph $G = (V, E, w)$, let \mathcal{A} be a c -fuzzy Δ -chop with respect to some root $r \in V$. \mathcal{A} is a τ -scattering chop if each shortest path of length at most Δ has at most τ edges cut by \mathcal{A} where we say that an edge is cut by \mathcal{A} if it has endpoints in different fuzzy annuli of \mathcal{A} .*

► **Definition 15** (τ -Scatter-Choppable Graphs). *A family of graphs \mathcal{G} is τ -scatter-choppable if there exists a constant $0 \leq c < 1$ such that for any $G \in \mathcal{G}$ and $\Delta \geq 1$ there is some τ -scattering and c -fuzzy Δ -chop \mathcal{A} with respect to some root.*

We will say that \mathcal{G} is deterministic poly-time τ -scatter-choppable if the above chop \mathcal{A} for each $G \in \mathcal{G}$ can be computed in deterministic poly-time.

Lastly, we conclude that to give an $O(1)$ -scattering partition – and therefore to give an $O(1)$ -SPR solution – for a K_h -minor-free graph family it suffices to show that such a family is $O(1)$ -scatter choppable.

► **Lemma 16.** *Fix a constant $h \geq 2$ and let \mathcal{G}_h be all K_h -minor-free graphs. Then, if \mathcal{G}_h is τ -scatter-choppable then every $G \in \mathcal{G}_h$ is $O(\tau^{h-1})$ -scatterable.*

Proof. The claim is almost immediate from Lemma 10 and the fact that all subpaths of a shortest path are themselves shortest paths.

In particular, first fix a sufficiently small constant c' to be chosen later. Then, consider a $G \in \mathcal{G}_h$ and fix a Δ . By assumption we know that G is τ -scatter-choppable and since each subgraph of G is in \mathcal{G}_h so too is each subgraph of G . Thus, we may let \mathcal{C} be the connected components resulting from $h - 1$ levels of c' -fuzzy and $(c'\Delta)$ -chops which are τ -scattering.

We claim that for sufficiently small c' we have that \mathcal{C} is a $(\frac{\tau^{h-1}}{c'}, \Delta)$ -scattering partition. By Lemma 10 the diameter of each part in \mathcal{C} is at most $O(c' \cdot h \cdot \Delta) \leq \Delta$ for sufficiently small c' . Next, consider a shortest path P of length at most Δ . We can partition the edges of P into at most $\frac{1}{c'}$ shortest paths P_1, P_2, \dots , each of length at most $c' \cdot \Delta$. Thus, it suffices to show that each P_i satisfies $|\{C \in \mathcal{C} : P_i \cap C \neq \emptyset\}| \leq \tau^{h-1}$.

We argue by induction on the number of levels of chops that after $h' < h$ chops we have $|\{C \in \mathcal{C} : P_i \cap C \neq \emptyset\}| \leq \tau^{h'}$. Suppose we perform just one chop; i.e. $h' = 1$. Then, since our chops are τ -scattering we know that P will be cut at most τ times and so be incident to at


most τ components of \mathcal{C} as required. Next, suppose we perform $h' > 1$ levels of chops. Then our top-level chop will partition the vertices of P_i into at most τ components. By induction and the fact that each subpath of P_i is itself a shortest path of length at most $c'\Delta$, we know that the vertices of P_i in each such component are broken into at most $\tau^{h'-1}$ components and so P_i will be incident to at most $\tau^{h'}$ components as required. As we perform $h - 1$ levels of chops, it follows that \mathcal{C} is indeed a $\left(\frac{\tau^{h-1}}{c'}, \Delta\right)$ -scattering partition. \blacktriangleleft

References

- 1 Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen Kobourov, and Richard Spence. Graph spanners: A tutorial review. *Computer Science Review*, 37:100253, 2020.
- 2 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.
- 3 Sanjeev Arora, Michelangelo Grigni, David R Karger, Philip N Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph tsp. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, volume 98, pages 33–41, 1998.
- 4 Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Symposium on Foundations of Computer Science (FOCS)*, pages 184–193. IEEE, 1996.
- 5 Amitabh Basu and Anupam Gupta. Steiner point removal in graph metrics, 2008. Unpublished Manuscript, available from <https://www.ams.jhu.edu/~abasu9/papers/SPR.pdf>.
- 6 Hans L Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1-2):1–45, 1998.
- 7 Bo Brinkman, Adriana Karagiozova, and James R Lee. Vertex cuts, random walks, and dimension reduction in series-parallel graphs. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 621–630, 2007.
- 8 T-H Hubert Chan, Donglin Xia, Goran Konjevod, and Andrea Richa. A tight lower bound for the steiner point removal problem on trees. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 70–81. Springer, 2006.
- 9 Chandra Chekuri, Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. Embedding k-outerplanar graphs into l_1 . *SIAM Journal on Discrete Mathematics*, 20(1):119–136, 2006.
- 10 Yun Kuen Cheung. Steiner point removal—distant terminals don’t (really) bother. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1353–1360. SIAM, 2018.
- 11 Yun Kuen Cheung, Gramoz Goranci, and Monika Henzinger. Graph minors for preserving terminal distances approximately—lower and upper bounds. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2016.
- 12 Richard J Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2):303–318, 1965.
- 13 Yuval Emek and David Peleg. A tight upper bound on the probabilistic embedding of series-parallel graphs. *SIAM Journal on Discrete Mathematics*, 23(4):1827–1841, 2010.
- 14 Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Racke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM Journal on Computing*, 43(4):1239–1262, 2014.
- 15 David Eppstein. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41–55, 1992.
- 16 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.
- 17 Arnold Filtser. Steiner point removal with distortion $o(\log k)$. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1361–1373. SIAM, 2018.

- 18 Arnold Filtser. Steiner point removal with distortion $o(\log k)$, using the noisy-voronoi algorithm. *arXiv preprint*, 2018. [arXiv:1808.02800](https://arxiv.org/abs/1808.02800).
- 19 Arnold Filtser. Scattering and sparse partitions, and their applications. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2020.
- 20 Arnold Filtser, Robert Krauthgamer, and Ohad Trabelsi. Relaxed voronoi: A simple framework for terminal-clustering problems. In *SIAM Symposium on Simplicity in Algorithms (SOSA)*, 2018.
- 21 Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. *SIAM Journal on Discrete Mathematics*, 34(1):130–162, 2020.
- 22 Anupam Gupta. Steiner points in tree metrics don't (really) help. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 682–690, 2001.
- 23 Anupam Gupta, Robert Krauthgamer, and James R Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Symposium on Foundations of Computer Science (FOCS)*, pages 534–543. IEEE, 2003.
- 24 Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. Cuts, trees and l_1 -embeddings of graphs. In *Symposium on Foundations of Computer Science (FOCS)*, pages 399–408. IEEE, 1999.
- 25 Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. Cuts, trees and l_1 -embeddings of graphs. *Combinatorica*, 24(2):233–269, 2004.
- 26 Frank Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, 1975.
- 27 Lior Kamma, Robert Krauthgamer, and Huy L Nguyen. Cutting corners cheaply, or how to remove steiner points. *SIAM Journal on Computing*, 44(4):975–995, 2015.
- 28 Samir Khuller. Ear decompositions. *SigAct News*, 20(1):128, 1989.
- 29 Philip Klein, Serge A Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 682–690, 1993.
- 30 Robert Krauthgamer, Huy L Nguyen, and Tamar Zondiner. Preserving terminal distances using minors. *SIAM Journal on Discrete Mathematics*, 28(1):127–141, 2014.
- 31 Robert Krauthgamer and Havana Inbal Rika. Refined vertex sparsifiers of planar graphs. *SIAM Journal on Discrete Mathematics*, 34(1):101–129, 2020.
- 32 James R. Lee and Cyrus Rashtchian. A simpler proof of the kpr theorem. <https://tcsmath.wordpress.com/tag/klein-plotkin-rao/>, January 2012.
- 33 Haruko Okamura and Paul D Seymour. Multicommodity flows in planar graphs. *Journal of Combinatorial Theory, Series B*, 31(1):75–81, 1981.
- 34 Neil Robertson and Paul D Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.
- 35 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 261–272. Springer, 2005.
- 36 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.
- 37 Thomas Victor Wimer. Linear algorithms on k -terminal graphs. *PhD Thesis*, 1987. AAI8803914.

Chromatic k -Nearest Neighbor Queries

Thijs van der Horst 

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Maarten Löffler  

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Frank Staals 

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

Let P be a set of n colored points. We develop efficient data structures that store P and can answer chromatic k -nearest neighbor (k -NN) queries. Such a query consists of a query point q and a number k , and asks for the color that appears most frequently among the k points in P closest to q . Answering such queries efficiently is the key to obtain fast k -NN classifiers. Our main aim is to obtain query times that are independent of k while using near-linear space.

We show that this is possible using a combination of two data structures. The first data structure allow us to compute a region containing exactly the k -nearest neighbors of a query point q , and the second data structure can then report the most frequent color in such a region. This leads to linear space data structures with query times of $O(n^{1/2} \log n)$ for points in \mathbb{R}^1 , and with query times varying between $O(n^{2/3} \log^{2/3} n)$ and $O(n^{5/6} \text{polylog } n)$, depending on the distance measure used, for points in \mathbb{R}^2 . These results can be extended to work in higher dimensions as well. Since the query times are still fairly large we also consider approximations. If we are allowed to report a color that appears at least $(1 - \varepsilon)f^*$ times, where f^* is the frequency of the most frequent color, we obtain a query time of $O(\log n + \log \log \frac{1}{1-\varepsilon} n)$ in \mathbb{R}^1 and expected query times ranging between $\tilde{O}(n^{1/2}\varepsilon^{-3/2})$ and $\tilde{O}(n^{1/2}\varepsilon^{-5/2})$ in \mathbb{R}^2 using near-linear space (ignoring polylogarithmic factors).

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases data structure, nearest neighbor, classification

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.67

Related Version *Full Version:* <https://doi.org/10.48550/arXiv.2205.00277>

Funding *Maarten Löffler:* Partially supported by the Dutch Research Council (NWO) under the project numbers 614.001.504 and 628.011.005.

Acknowledgements We would like to thank an anonymous reviewer for the randomized solution presented in Section 3.2.1, which led to our current solution for finding $\mathcal{D}_2^k(q)$ in Section 3.2.2.

1 Introduction

One of the most popular approaches for classification problems is to use a k -Nearest Neighbor (k -NN) classifier [3, 10, 12, 14]. In a k -NN classifier the predicted class of a query item q is taken to be the most frequently appearing class among the k items most similar to q . One can model this as a geometric problem in which the input items are represented by a set P of n colored points in \mathbb{R}^d : the color of the points represents their class, and the distance between points measures their similarity. The goal is then to store P so that one can efficiently find the color (class) c^* most frequently occurring among the k points in P closest to a query point q . See Figure 1(left). We refer to such queries as *chromatic k -NN queries*. To answer such queries, k -NN classifiers often store P in, e.g., a kd-tree and answer queries by explicitly reporting the k points closest to q , scanning through this set to compute the most frequently occurring color [3]. Unfortunately, for many distance measures (including



© Thijs van der Horst, Maarten Löffler, and Frank Staals;
licensed under Creative Commons License CC-BY 4.0

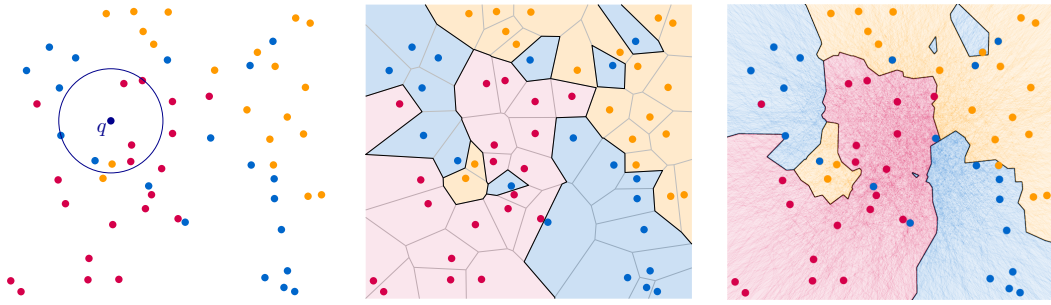
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 67; pp. 67:1–67:14

Leibniz International Proceedings in Informatics



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



■ **Figure 1** (left) A set of input points from three different classes (colors). The class of a query point q is determined by the labels of its k nearest neighbors (with $k = 7$ as shown here q is classified as red). (middle) The color partition for $k = 1$. (right) The color partition for $k = 3$.

the Euclidean distance) such an approach has no guarantees on the query time other than the trivial $O(n)$ time bound. Even assuming that the dependency on n during the query time is small (e.g. when the points are nicely distributed [11]), the approach requires $\Theta(k)$ time to explicitly process all k points closest to q , whereas the desired output is only a single value: the most frequently appearing color. Hence, our main goal is to design a data structure to store P that has sublinear query time in terms of both n and k , while still using only small space. We focus our attention on the L_2 (Euclidean) distance, and the L_∞ distance metrics. Most of our ideas extend to more general distance measures and higher dimensions as well. However, already in the restricted settings presented here, designing data structures that provide guarantees on the space and query time turns out to be a challenging task.

If the value k is fixed in advance, one possible solution is to build the k^{th} -order Voronoi diagram $Vor_k(P)$ of P , and preprocess it for point location queries. The k^{th} -order Voronoi diagram is a partition of \mathbb{R}^d into maximal cells for which all points in a cell (a *Voronoi region*) $V_{k,P}(S)$ have the same set S of k closest points from P . Hence, in each Voronoi region there is a fixed color that occurs most frequently among S . See Figure 1(right) for an illustration. By storing $Vor_k(P)$ in a data structure for efficient (i.e. $O(\log n)$ time) point location queries we can also answer chromatic k -NN queries efficiently. However, unfortunately $Vor_k(P)$ may have size $\Theta(k(n-k))$ [15] (for points in \mathbb{R}^2 and the L_2 distance). For other L_m distances the diagram is similarly large [17, 6]. Hence, we are interested in solutions that use less, preferably near-linear, space.

The only result on the theory of chromatic k -NN queries that we are aware of is that of Mount et al. [20]. They study the problem in the case that we measure distance using the Euclidean metric and that the number of colors c , as well as the parameter k , are small constants. Mount et al. state that it is unclear how to obtain a query time independent of k , and instead analyze the query times in terms of the *chromatic density* ρ of a query q . The chromatic density is a term depending on the distance from the query point q to the k^{th} nearest neighbor of q , and the distance from q to the first point at which the answer of the query would change (e.g. the $(k+t)^{\text{th}}$ nearest neighbor of q for some $t > 0$). The intuition is that if many points near q have the same color, queries should be easier to answer than when there are multiple colors with roughly the same number of points. The chromatic density term models this. Their main result is a linear space data structure for points in \mathbb{R}^d that supports $O(\log^2 n + (1/\rho)^d \log(1/\rho))$ time queries. We aim for bounds only in terms of combinatorial properties (i.e. n , c , and k) and allow the number of colors, as well as the parameter k , to be comparable to n . Our results are particularly relevant when k and c are large compared to n .

■ **Table 1** Our results for exact chromatic k -nearest neighbors problems. Bounds marked with * are expected bounds. The general L_m metric bounds hold for $m = O(1)$.

Dimension	Metric	Preprocessing time	Space	Query time
$d = 1$	L_m	$O(n^{3/2} \log n)$	$O(n)$	$O(n^{1/2} \log n)$
$d = 2$	L_m	$\tilde{O}(n^{2/(4+\delta)})$	$O(n)$	$\tilde{O}(n^{1-1/(12+3\delta)})$
$d \geq 3$	L_m	$\tilde{O}(n^{2-1/d} + n^{1+d/(2d-2+\delta)})$	$O(n)$	$\tilde{O}(n^{1-1/((d+1)(2d-2+\delta))})$
$d \geq 2$	L_∞	$O(n^{1+d/(d+1)})$	$O(n)$	$O(n^{1-1/(d+1)+\delta})$
		$\tilde{O}(n^{1+d/(d+1)})$	$O(n \log^{d-1} n)$	$O((n \log^{d-1} n)^{1-1/(d+1)})$
	L_2	$\tilde{O}(n^{2-1/d} + n^{1+d/(d+1)})^*$	$O(n)$	$\tilde{O}(n^{1-(d-1)/d(d+1)})$

■ **Table 2** Our results for the approximate chromatic k -nearest neighbors problems.

Dimension	Metric	Preprocessing time	Space	Query time
$d = 1$	L_m	$O(n \log_{\frac{1}{1-\varepsilon}} n)$	$O(n\varepsilon^{-1})$	$O(\log n + \log \log_{\frac{1}{1-\varepsilon}} n)$
$d = 2$	L_∞	$\tilde{O}(n\varepsilon^{-6})^*$	$\tilde{O}(n\varepsilon^{-6})$	$\tilde{O}(n^{1/2}\varepsilon^{-5/2})^*$
	L_2	$\tilde{O}(n^{1+\delta} + n\varepsilon^{-7})^*$	$\tilde{O}(n\varepsilon^{-4})$	$\tilde{O}(n^{1/2}\varepsilon^{-3/2})^*$

Our approach. Our main idea is to answer a query in two steps. (1) We identify a region $\mathcal{D}_m^k(q)$ that contains exactly the set $k\text{-NN}_m(q)$ of the k sites closest to q according to distance metric L_m . (2) We then find the *mode color* c^* ; that is, the most frequently occurring color among the points in the region $\mathcal{D}_m^k(q)$. This way, we never have to explicitly enumerate the set $k\text{-NN}_m(q)$. We will design separate data structures for these two steps. Our data structure for step (1) will find the smallest metric disk $\mathcal{D}_m^k(q)$ containing $k\text{-NN}_m(q)$ centered at q . We refer to such a query as a *range finding* query. If the distance used is clear from the context we may write $k\text{-NN}(q)$ and $\mathcal{D}^k(q)$ instead.

Range mode queries. The data structure in step (2) answers so-called *range mode* queries. For these data structures we exploit and extend the result of Chan et al. [8]. They show an array A with n entries can be stored in a linear space data structure that allows reporting the mode of a query range (interval) $A[i..j]$ in $O(n^{1/2})$ time. Furthermore, points in \mathbb{R}^d can be stored in $O(n \text{polylog } n + r^{2d})$ space so that range mode queries with axis-aligned orthogonal ranges can be answered in $O((n/r) \text{polylog } n)$ time. Here, $r \in [1, n]$ is a user-choosable parameter. In particular, setting $r = \lceil n^{1/2d} \rceil$ yields an $O(n \text{polylog } n)$ space solution with $O(n^{1-1/2d} \text{polylog } n)$ query time. Range mode queries with halfspaces can be answered in $O((n/r)^{1-1/d^2} + \text{polylog } n)$ time using $O(nr^{d-1})$ space [8]. Range mode queries in arrays have also been considered in an approximate setting [7]. The goal is then to report an element that appears sufficiently often in the range.

Results and organization. Refer to Table 1 for an overview of our exact solutions. We first consider the problem for n points in \mathbb{R}^1 . In this setting, we develop an optimal linear space data structure that can find $\mathcal{D}_m^k(q)$ in $O(\log n)$ time, for any $m \geq 1$ (Section 2). We then use Chan et al. [8]’s data structure to report the mode color in $\mathcal{D}_m^k(q)$. Since we present all of our data structures in the pointer machine model augmented with real-valued arithmetic, this

yields an $O(n^{1/2} \log n)$ time query algorithm. There is a conditional $\Omega(n^{1/2-\delta})$ lowerbound for chromatic k -NN queries using linear space and $O(n^{3/2})$ preprocessing time (refer to Section 5) so this result is likely near-optimal. In Section 3 we present our data structure for finding $\mathcal{D}_m^k(q)$ in \mathbb{R}^2 . For the L_∞ metric we show that we can essentially find $\mathcal{D}_m^k(q)$ using a binary search on the radius of the disk, and thus there is a simple $O(n \log n)$ size data structure that allows us to find the range $\mathcal{D}^k(q)$ in $O(\log^2 n)$ time (or $O(n^\delta)$ time, for an arbitrarily small $\delta > 0$, in case of a linear space structure). For the L_2 metric, we can no longer easily access a discrete set of candidate radii. It is tempting to therefore replace the binary search by a parametric search [18, 19]. However, the basic such approach squares the $\tilde{O}(n^{1/2})$ time required to solve the decision problem and is thus not applicable. The full strategy requires a way to generate independent comparisons; typically by designing a parallel decision algorithm [18]. Neither task is straightforward to achieve. Instead, we show that we can directly adapt the query algorithm for answering semi-algebraic range queries [2] (essentially the decision algorithm in the approach sketched above) to find $\mathcal{D}_2^k(q)$ in $O(n^{1/2} \text{polylog } n)$ time. Unfortunately, the final query time for chromatic k -NN queries is dominated by the $O(n^{5/6} \text{polylog } n)$ time range mode queries, for which we use a slight variation of the data structure of Chan et al. [8]. We briefly discuss these results in Section 4. We do show that the data structure can be constructed in $O(n^{5/3})$ expected time, rather than the straightforward $O(n^2)$ time implementation that directly follows from the description of Chan et al.. For the L_∞ metric we can answer range mode queries in $O(n^{3/4} \text{polylog } n)$ time using Chan et al.'s data structure for orthogonal ranges. However, we show that we can exploit that the ranges are squares and use a cutting-based approach (similar to the one used for the L_2 distance) to answer queries in $O(n^{2/3} \log^{2/3} n)$ time instead. If we wish to reduce the space from $O(n \log n)$ to linear the query time becomes $O(n^{2/3+\delta})$.

Since the query times are still rather large, we then turn our attention to approximations; refer to Table 2 for an overview. If f^* is the frequency of the mode color among k -NN(q) then our data structures may return a color that appears at least $(1 - \varepsilon)f^*$ times. In case of the L_2 distance our data structure presented in Section 6 now achieves roughly $\tilde{O}(n^{1/2} \varepsilon^{-3/2})$ query time, where the \tilde{O} notation hides polylogarithmic factors of n and ε . The main idea is that approximate levels in the arrangement of distance functions have relatively low complexity, and thus we can store them to efficiently answer approximate range mode queries.

2 The one-dimensional problem

In this section we consider the case where P is a set of n points in \mathbb{R}^1 . In this case all L_m -distance metrics with $m \geq 1$ are the same, i.e. $L_m(a, b) = |a - b|$. We develop a linear space data structure supporting queries in $O(n^{1/2} \log n)$ time. Building the data structure will take $O(n^{3/2})$ time. We follow the general two-step approach sketched in Section 1. We first show that there is an optimal, linear-space data structure with which we can find $\mathcal{D}^k(q)$ in $O(\log n)$ time, even if k is part of the query. As we then briefly describe how we can directly use the data structure by Chan et al. [8] to find the mode color of the points in $\mathcal{D}^k(q)$ in $O(n^{1/2} \log n)$ time.

Range finding queries. Given a set P of n points in \mathbb{R}^1 , we wish to store P so that given a query, consisting of a point $q \in \mathbb{R}^1$ and a natural number k , we can efficiently find the k^{th} furthest point from q , and thus $\mathcal{D}_m^k(q)$. We show that by storing P in sorted order in an appropriate binary search tree, we can answer such queries in $O(\log n)$ time. To this end, we first consider answering so called rank queries on a ordered set, represented by a pair of binary search trees. This turns out to be the crucial ingredient in the data structure.

Let $R \cup B$ be an ordered set of elements, and let T_R be a binary search tree whose internal nodes store the elements from R , and in which each node is annotated with the number of elements in that subtree. Similarly, let T_B be a binary search tree storing B . We can efficiently compute the element among $R \cup B$ with rank k (i.e. the k^{th} smallest element):

► **Lemma 2.1.** *Let T_B and T_R be two binary search trees with size annotations, and let k be a natural number. We can compute the element of rank k among $B \cup R$ in $O(\text{height}(T_B) + \text{height}(T_R))$ time.*

To support efficiently querying $\mathcal{D}^k(q)$ we now store P in a balanced binary search tree T_P with subtree-size annotations, so that we can: (i) search for the element of rank k , and (ii) given a query value $q \in \mathbb{R}^1$ we can split the tree at q in $O(\log n)$ time. We can implement T_P using e.g. a red black tree [22] (although with some care even a simple static balanced binary search tree will suffice). We will use the split operations only to answer queries; so we use path copying in this operation, so that we can still access the original tree once a query finishes [21]. The data structure uses $O(n)$ space, and can be built in $O(n \log n)$ time.

Given a query (q, k) the main idea is now to split T_P into two trees $T_{P^<}$ and $T_{P^{\geq}}$, where $P^< \subseteq P$ is the set of points left of q and $P^{\geq} = P \setminus P^<$ is the remaining set of points right of q (or coinciding with q). Observe that in these two trees, the points are actually ordered by distance to q (albeit for $T_{P^<}$ the points are stored in decreasing order while the points in $T_{P^{\geq}}$ are stored in increasing order). So, we can essentially use the procedure from Lemma 2.1 on the trees $T_{P^<}$ and $T_{P^{\geq}}$ to find the point in $P^{\leq} \cup P^>$ with rank k (according to the “by distance to q ”-order). The only difference with the algorithm as described in Lemma 2.1 is that for $T_{P^<}$ the roles of the left and right subtree are reversed. Splitting T into $T_{P^<}$ and $T_{P^{\geq}}$ takes $O(\log n)$ time. Since both subtrees have height at most $O(\log n)$, Lemma 2.1 also takes $O(\log n)$ time. So, we obtain the following result:

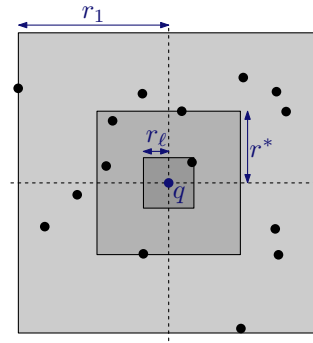
► **Theorem 2.2.** *Let P be a set of n points in \mathbb{R}^1 . In $O(n \log n)$ time we can build a linear space data structure so that given a query q, k we can find the smallest disk $\mathcal{D}_m^k(q)$, with respect to any L_m metric, containing k - $\text{NN}_m(q)$ in $O(\log n)$ time.*

Range mode queries. What remains is to store P such that given a query interval Q we can efficiently report the mode color among $P \cap Q$. Chan et al. [8] show that an array A of size n can be preprocessed in $O(n^{3/2})$ time into a linear space structure that can report the range mode of a query range $A[i : j]$ in $O(n^{1/2})$ time. By implementing arrays with balanced binary search trees, we can also implement this structure in the pointer machine model. This increases the query and preprocessing times by an $O(\log n)$ factor. We then store (the colors of) the points in increasing order in this structure. Together with Theorem 2.2 we obtain:

► **Theorem 2.3.** *Let P be a set of n points in \mathbb{R}^1 . In $O(n^{3/2} \log n)$ time, we can build a data structure of size $O(n)$, that answers chromatic k - NN queries on P in $O(n^{1/2} \log n)$ time.*

3 Range finding queries two dimensions

In this section we give a data structure that, given a query point $q \in \mathbb{R}^2$, reports the smallest range $\mathcal{D}_m^k(q)$ centered at q containing k - $\text{NN}_m(q)$. In Section 3.1 we consider the case where the L_∞ metric is used. In Section 3.2 we then consider the L_2 metric.



■ **Figure 2** The considered radii r_i , and r^* , for $k = 5$.

3.1 The L_∞ metric case

For a given value $r \geq 0$, define $S(q, r) = [q_x - r, q_x + r] \times [q_y - r, q_y + r]$ as the axis-aligned square with sidelength $2r$ centered at q . We call r the *radius* of such a square. Now observe that $\mathcal{D}^k(q) = S(q, r^*)$ is also an axis-aligned square, in particular with radius r^* equal to the distance $L_\infty(q, p)$ between q and the k^{th} nearest neighbor p of q . Thus we either have $r^* = |q_x - p_x|$ or $r^* = |q_y - p_y|$. This leads us to the following data structure. We store the x -coordinates x_1, \dots, x_n of the points in P in increasing order in a balanced binary search tree. Similarly, we store the y -coordinates of the points in P in sorted order y_1, \dots, y_n . We set $x_0 = y_0 = -\infty$ and $x_{n+1} = y_{n+1} = \infty$, and call these values coordinates as well. In addition, we store P in a data structure for $O(\log n)$ time orthogonal range counting queries, for which we use a range tree [5, 23]. The entire data structure can be constructed in $O(n \log n)$ time, and uses $O(n \log n)$ space.

Now let x_0, \dots, x_ℓ be the x -coordinates that are at most q_x . The sequence $r_i = |q_x - x_i|$, for $i = 0, \dots, \ell$, defines a sequence of decreasing radii. See Figure 2 for an illustration. We can find the smallest radius r_i for which $S(q, r_i)$ contains at least k points by performing binary search over the radii, performing orthogonal range counting at each step to guide the search. By performing a similar procedure for the x -coordinates greater than q_x , as well as for the y -coordinates, we obtain a set of four squares, that each contain at least k points. The smallest of these squares contains exactly k points and is thus $\mathcal{D}^k(q)$. As each procedure performs $O(\log n)$ orthogonal range counting queries, we obtain the following theorem.

► **Theorem 3.1.** *Let P be a set of n points in \mathbb{R}^2 . In $O(n \log n)$ time, we can build a data structure of size $O(n \log n)$, that can report $\mathcal{D}_1^k(q)$ and $\mathcal{D}_\infty^k(q)$ in $O(\log^2 n)$ time.*

Following an idea of Chan et al. [8] we can reduce the space used by replacing the binary range tree used for the orthogonal range queries by one with fanout n^δ for some constant $\delta > 0$. This increases the query time to $O(n^\delta + k')$ time, where k' is the output size.

► **Theorem 3.2.** *Let P be a set of n points in \mathbb{R}^2 . Let $\delta > 0$ be an arbitrarily small constant. In $O(n \log n)$ time, we can build a data structure of $O(n)$ size, that can report $\mathcal{D}_1^k(q)$ and $\mathcal{D}_\infty^k(q)$ in $O(n^\delta)$ time.*

3.2 The L_2 metric case

3.2.1 Randomized query times

In this section, we sketch a simple randomized data structure that finds $\mathcal{D}_2^k(q)$ in expected $O(n^{1/2} \log^{B+1} n)$ time¹. The data structure consists of the large fan-out partition tree of Agarwal et al. [2], used for semialgebraic range searching. Together with this tree, we take a random sample P' of P by including each point with probability $1/n^{1/2}$. Note that this random sample will contain $n^{1/2}$ points in expectation.

The main idea is to combine binary search on the ordered distances $R = \{L_2(q, p') \mid p' \in P'\}$ with circular range counting. Let r^* be the distance between q and its k^{th} nearest neighbor among P . We then search for two consecutive distances $r_i, r_{i+1} \in R$, such that $r_i \leq r^* \leq r_{i+1}$. Because the number of points $p \in P$ with $r_i \leq L_2(q, p) \leq r_{i+1}$ is $n^{1/2}$ in expectation, we can then afford to report these points with semialgebraic range reporting, and combining binary search with range counting again to find r^* . This leads to an expected query time of $O(n^{1/2} \log^{B+1} n)$.

3.2.2 Worst-case query times

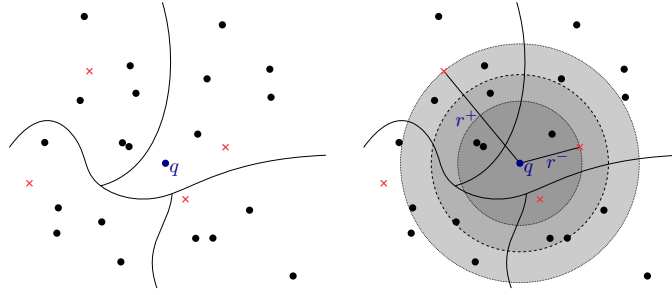
In this section we show how to achieve the same complexity bounds of Section 3.2.1, but with a worst-case query time bound, rather than a randomized bound.

The data structure. For now, we assume that the set P lies in D_0 -general position for some constant D_0 (see [2] for a definition). The details on this assumption are not important, and we show in the full version of the paper how to handle arbitrary point sets. The data structure consists of two copies of the large fan-out (fan-out n^δ , for some constant $\delta > 0$) partition tree of Agarwal et al. [2], built on P . The first copy, which we call \mathcal{T} , will be augmented slightly to support generating candidate ranges (disks) that will eventually lead to $\mathcal{D}^k(q)$. The second copy will be used as a black box, answering circular range counting queries to guide the search for $\mathcal{D}^k(q)$ by counting the number of points inside the candidate ranges.

The tree \mathcal{T} is constructed by recursively partitioning the space into open, connected regions, called *cells*. Once a cell contains a small (constant) number of points of P , the recursion stops and \mathcal{T} gets a leaf node containing these points. There may be points of P that do not lie on these cells, but rather on the zero set of the partitioning polynomial used to partition the space. For range searching with arbitrary point sets, Agarwal et al. [2] store these points in an auxiliary data structure. However, with our assumption that P lies in D_0 -general position, we do not need this auxiliary data structure, and will simply store the points inside a leaf node, whose parent is the node corresponding to the partitioning polynomial. We further adjust \mathcal{T} such that each internal node corresponding to a cell ω stores an arbitrary point in ω . During the construction of \mathcal{T} , a point inside each cell is already computed. Hence the construction time is unaltered.

Answering a query. To query the structure with a query point q , we keep track of a set of nodes N_i for each level i of \mathcal{T} that is explored by our algorithm. With slight abuse of terminology, we refer to the sets P' of points stored in the leaves of \mathcal{T} as cells. Let Ω_i denote

¹ We would like to thank an anonymous reviewer for the solution sketched in this section, which led to our current solution for finding $\mathcal{D}_2^k(q)$ in Section 3.2.



■ **Figure 3** (left) A partitioning of P into four cells. The red crosses are the points $p(\omega)$. (right) The disk $\mathcal{D}^k(q)$ (dashed boundary) and the disks $D(q, r^-)$ (dotted boundary, dark gray) and $D(q, r^+)$ (dotted boundary, light gray).

the cells corresponding to the internal nodes in N_i , and P_i denote the cells corresponding to the leaf nodes in N_i . We maintain that $p^k(q)$, the k^{th} nearest neighbor of q , is contained in a cell in $\Omega_i \cup P_i$. Initially, N_1 contains the root node. If \mathcal{T} is a single leaf, then the cell corresponding to the root node will be stored in P_1 . Otherwise, it will be stored in Ω_1 .

The query algorithm works as follows. Say the algorithm is at level i in \mathcal{T} . For a cell $\omega \in \Omega_i$ stored in an internal node, let $p(\omega)$ be the point inside ω that was stored with it. Let

$$R_i = \{L_2(q, p(\omega)) \mid \omega \in \Omega_i\} \cup \bigcup_{P'_i \in P_i} \{L_2(q, p) \mid p \in P'_i\}$$

be the set of distances to these points $p(\omega)$, as well as to all points stored in the leaves in P_i and in the nodes in N_i . Let r^* be the distance between q and its k^{th} nearest neighbor among P . This is the radius of $\mathcal{D}^k(q)$. To find this radius, we compute the largest distance $r^- \in R_i$, and the smallest distance $r^+ \in R_i$, such that $r^- < r^* \leq r^+$. See Figure 3. If r^- (respectively r^+) does not exist, set it to 0 (respectively ∞). We show how to compute r^+ . Computing r^- works similarly.

To compute r^+ , we use a combination of median finding and binary search. For some radius $r \in R_i$, we decide if $r^+ > r$ or $r^+ \leq r$ using a circular counting query. If $D(q, r)$ contains less than k points, we have $r^+ > r$. Otherwise, we have $r^+ \leq r$. By performing this procedure for the median radius in R_i , we can discard half of R_i with each query.

Once we have that r^+ is the distance between q and a point in P (it is constructed through a leaf node of \mathcal{T}), we have found r^* . We then terminate the algorithm, returning the disk with the found radius. Otherwise we continue the search in the next level of \mathcal{T} . To continue the search through the tree, we construct the set N_{i+1} by replacing every node $\nu \in N_i$ with its children whose cells are crossed by one of $D(q, r^-)$ and $D(q, r^+)$. A cell ω is *crossed* by a disk D if $\omega \cap D \neq \emptyset$ and $\omega \not\subseteq D$. The sets Ω_{i+1} and P_{i+1} are then constructed from these child nodes. Once these sets are constructed, we advance the algorithm to level $i+1$ and repeat the procedure.

► **Lemma 3.3.** *The query algorithm correctly returns $\mathcal{D}^k(q)$.*

Proof. We claim that this algorithm correctly returns $\mathcal{D}^k(q)$. First, note that if the algorithm returns a disk, that disk contains k points, and its radius is equal to the distance between q and a point of P . Therefore, it is indeed $\mathcal{D}^k(q)$. We now show that our algorithm will always return a disk, and therefore that our algorithm is correct. To show this, it suffices to show that for every level i of \mathcal{T} traversed by our algorithm, the set $\Omega_i \cup P_i$ contains the cell containing $p^k(q)$.

We give a proof by induction. It holds trivially that $\Omega_1 \cup P_1$ contains a cell containing $p^k(q)$. We prove that if $\Omega_i \cup P_i$ contains a cell ω' containing $p^k(q)$, then either the algorithm terminates and returns $\mathcal{D}^k(q)$, or there is a cell $\omega \in \Omega_{i+1} \cup P_{i+1}$ containing $p^k(q)$.

If $\omega' \in P_i$, then R_i will contain r^* . It will then find $r^+ = r^*$ and terminate, returning $D(q, r^+) = \mathcal{D}^k(q)$. Now assume that $\omega' \in \Omega_i$. Let $\nu \in N_i$ be the internal node corresponding to ω' . Now let ω be the cell stored in a child of ν , such that $p^k(q)$ lies in ω . We show that $\omega \in \Omega_{i+1} \cup P_{i+1}$.

Our algorithm performs repeated median finding on the radii in R_i , resulting in the largest radius $r^- \in R_i$ and smallest radius $r^+ \in R_i$, such that $D(q, r^-)$ and $D(q, r^+)$, contains less than, respectively at least, k points of P . Let r_ω be the distance between q and $p(\omega)$, the point in ω that was stored in \mathcal{T} . If $r_\omega < r^*$, then we have that $r_\omega \leq r^-$, implying that $D(q, r^-)$ intersects ω . Also, because ω is open, and because $p^k(q) \in \omega$, there must be a point $p' \in \omega$ such that $r^- < r^* = L_2(q, p^k(q)) < L_2(q, p')$. This shows that ω is not contained in $D(q, r^-)$, and thus that $D(q, r^-)$ crosses ω . With similar reasoning, it can be seen that if $r_\omega \geq r^*$, then $D(q, r^+)$ crosses ω . Thus, ω will be crossed by at least one of $D(q, r^-)$ and $D(q, r^+)$, and thus $\omega \in \Omega_{i+1} \cup P_{i+1}$. Hence, our algorithm is correct. ◀

► **Theorem 3.4.** *Let P be a set of n points in \mathbb{R}^2 . Let $\delta > 0$ be an arbitrarily small constant. In $O(n^{1+\delta})$ expected time, we can build a data structure of $O(n)$ size, that can report $\mathcal{D}_2^k(q)$ in $O(n^{1/2} \text{polylog } n)$ time.*

4 Range mode queries in two dimensions

In this section we discuss answering range mode queries in \mathbb{R}^2 , and see how we can use them together with the data structures from Section 3 to answer chromatic k -NN queries. Our results build on the data structure of Chan et al. [8] for finding the mode color among three-dimensional points in halfspaces. We show that using a standard lifting transformation that maps the points $P \subseteq \mathbb{R}^2$ into planes in \mathbb{R}^3 , we can apply their result to answer range mode queries with disks in the L_2 metric. Our main contribution is that we show that a similar approach can answer range mode queries with squares (disks in the L_∞ metric). Somewhat surprisingly, this leads to better query times compared to directly using the existing range mode data structures for orthogonal ranges [8]. Finally, we show that we can, in fact, construct the data structures in (expected) $O(n^{5/3})$ time rather than $O(n^2)$ (worst-case) time. We briefly sketch these ideas here. Refer to the full version for details.

The data structure for the L_2 metric. We lift the set of points $P \subseteq \mathbb{R}^2$ to a set of planes H in \mathbb{R}^3 . The points in a query disk Q map to the subset of planes passing below a point h^* . Hence, we have to report the mode color c of these planes. The main idea in the data structure of Chan et al. [8] is to build a $(1/r)$ -cutting on the planes in H ; a subdivision of \mathbb{R}^3 into $O(r^3)$ interiorly disjoint cells, whose interiors are each crossed by at most n/r planes (for $r = n^{1/3}$). Let Δ be the cell containing the query point h^* . The key insight is that the mode color c of the planes passing below the query point h^* is either the mode color c_Δ of all planes passing below Δ , or the color of one of the planes that intersect Δ . The data structure stores the color c_Δ for each cell Δ , so at query time we only have to consider the at most n/r colors of the planes that intersect Δ . Our insight is that for k -NN queries we can use the tools from Section 3 to find these colors in $\tilde{O}(n^{1/2})$ time, leading to a total query time of $\tilde{O}(n^{5/6})$ (rather than $\tilde{O}(n^{8/9})$ time in the case of arbitrary query points in \mathbb{R}^3).

The data structure for the L_∞ metric. For a point $p \in P \subseteq \mathbb{R}^2$, the graph of the distance function $L_\infty(p, q) = \max\{|p_x - q_x|, |p_y - q_y|\}$ forms an upside-down pyramid ∇_p in \mathbb{R}^3 . For a point $q \in \mathbb{R}^2$ we have that $L_\infty(p, q) \leq r$ if and only if (q_x, q_y, r) lies above ∇_p . Thus, we can again transform the problem to that of finding the mode color among those graphs in \mathbb{R}^3 that lie below a given query point. Using a similar cutting-based solution as sketched above – again exploiting that we can quickly find the (colors of the) graphs intersecting a cell Δ in \mathbb{R}^2 rather than \mathbb{R}^3 – yields a linear space data structure answering queries in an $O(n^{2/3+\delta})$ time. Using $O(n \log n)$ space we can decrease the query time to $O(n^{5/3} \log^{2/3} n)$.

Construction. Our final contribution in terms of the range mode data structures is that we show how to efficiently construct the above data structures. In case of both the L_2 and the L_∞ metrics, there is a somewhat straightforward algorithm to construct the above data structures in $O(n^2)$ time. The bottleneck being the time required to compute the mode color c_Δ for all cells Δ of the cutting. We argue that this can actually be done in roughly $\tilde{O}(nr^2 + r^3 n^\alpha)$ time, for some $\alpha \in (0, 1)$. For our choice of parameter r , this leads to algorithms with subquadratic running time. We summarize our results for the chromatic k -NN problem in the following theorems.

► **Theorem 4.1.** *Let P be a set of n points in \mathbb{R}^2 . There is a linear space data structure that answers chromatic k -NN queries on P with respect to the L_2 metric. Building the data structure takes expected $O(n^{5/3})$ time, and queries take $O(n^{5/6} \text{polylog } n)$ time.*

► **Theorem 4.2.** *Let P be a set of n points in \mathbb{R}^2 . There is a linear space data structure that answers chromatic k -NN queries on P with respect to the L_∞ metric. Building the data structure takes $O(n^{5/3})$ time, and queries take $O(n^{2/3+\delta})$ time. We can decrease the query time to $O(n^{2/3} \log^{2/3} n)$ time using $O(n \log n)$ space and $O(n^{5/3} \log^{2/3} n)$ preprocessing time.*

5 Lower bounds

We now discuss to what extent our results for the exact version of the problem may be improved further. Chan et al. [8] show that there is a conditional $\Omega(n^{1/2-\delta})$ time lower bound on range mode queries, provided we insist on using only linear space and $O(n^{3/2})$ preprocessing time. We extend this bound to chromatic k -NN queries in \mathbb{R}^1 . More generally, Lemma 5.1 shows that we can reduce chromatic k -NN queries in \mathbb{R}^d to range mode queries using range counting. Due to space restrictions, further details on the lower bound can be found in the full version of the paper.

► **Lemma 5.1.** *Let P be a set of n points in \mathbb{R}^d . A range mode query with a query ball \mathcal{D}_m (with respect to the L_m metric) can be answered using a single range counting query with query range \mathcal{D}_m and a single chromatic k -NN query.*

Proof. We use the range counting query to find the number of points in the range \mathcal{D}_m . Let this be k , and let q be the center point of the ball \mathcal{D}_m . Hence, $\mathcal{D}_m^k(q) = \mathcal{D}_m$, and thus the answer to the chromatic k -NN query with center q and value k is the mode color of \mathcal{D}_m . ◀

Relations to range counting queries. Next, we relate the cost of range finding queries, i.e. the problem solved in our first step, to range counting queries. Given a data structure for range finding queries we can answer range counting queries using only logarithmic overhead:

► **Lemma 5.2.** *Let P be a set of n points in \mathbb{R}^d . A range counting query on P with a disk \mathcal{D}_m under metric m can be performed using $O(\log n)$ range finding queries.*

Proof. Let q be the center of the query disk. We binary search over the integers $0, \dots, n$, using a range finding query to find a disk $\mathcal{D}_m^k(q)$ for each considered integer k . If the reported disk is smaller than \mathcal{D}_m , the number of points inside \mathcal{D}_m is at least k . Otherwise, the number of points is smaller than k . It follows that with $O(\log n)$ range finding queries, we can count the number of points in \mathcal{D}_m . ◀

It thus follows that range finding is roughly as difficult as range counting. In particular, a $Q(n)$ time lower bound for range counting queries using $S(n)$ space implies an $\tilde{\Omega}(Q(n))$ time lower bound for range finding queries with $S(n)$ space. For example, in the semigroup model there is an $\tilde{\Omega}(n/S(n)^{1/d})$ time lower bound for halfspace range counting [4]. Since every halfspace is a disk \mathcal{D}_2 (of radius ∞), this lower bound also holds for range counting with disks in the L_2 metric, and thus also for range finding.

The range mode queries from step 2 are also related to a form of range counting. A “type-2” range counting query with query range Q asks for all the distinct colors appearing in Q together with their frequencies, i.e. for each reported color c we must also report the number of points in $P \cap Q$ that have color c [9]. Clearly, answering “type-2” queries is more difficult than range counting (just assign all points the same color), so the above lower bounds also hold for “type-2” queries. Such “type-2” queries however also allow us to solve the range mode problem. When the number of colors is small (e.g. two), and we already know the number of points k in the query range $\mathcal{D}^k(q)$ it seems that answering range mode queries is not much easier than answering (“type-2”) range counting queries. We therefore conjecture that answering range mode queries is roughly as difficult as answering range counting queries.

► **Conjecture 5.3.** *If answering a range counting query with a query range \mathcal{D} using $S(n)$ space requires $Q(n)$ time then answering a range mode query with query range \mathcal{D} using $S(n)$ space requires $\tilde{\Omega}(Q(n))$ time.*

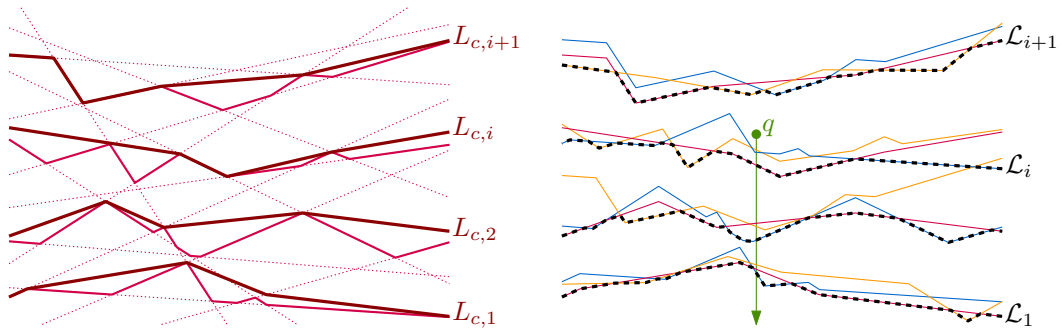
Note that this conjecture together with Lemma 5.1 would imply that answering a k -NN query is at least as hard as answering a range counting query. Furthermore, since we can answer a range counting query using $O(\log n)$ range finding queries (Lemma 5.2) that would then mean our two-step approach has negligible overhead with respect to an optimal solution to chromatic k -NN queries.

6 The approximate problems

In this section we sketch our approach for answering ε -approximate chromatic k -NN queries. Refer to the full version for details. Our goal is to report a color c that occurs at least $(1 - \varepsilon)f^*$ times, where f^* is the frequency of the mode color c^* of k -NN(q). We again use the two-step approach of finding the range $\mathcal{D}^k(q)$ (step (1)) and computing the mode of the range (step (2)). We use exact range finding data structures from Sections 2 and 3, and focus our attention on approximating step (2) for two reasons: first, the running times in our exact solutions are dominated by step (2), and second, it is unclear how to use approximate solutions to k -NN queries (that is, approximate ranges) and still obtain guarantees on the approximation factor of our ε -approximate chromatic k -NN queries.

For points in \mathbb{R}^1 we can directly use the result of Bose et al. [7] to answer $(1 - \varepsilon)$ -approximate range mode queries. In $O(n \log_{\frac{1}{1-\varepsilon}} n)$ time, we can thus build an $O(n/\varepsilon)$ size data structure that can answer k -NN queries in $O(\log n + \log \log_{\frac{1}{1-\varepsilon}} n)$ time.

For points in \mathbb{R}^2 , and the L_2 -metric, we answer approximate range mode queries as follows; we use similar ideas for the L_∞ metric. We use the standard lifting transformation to transform the set of points P a set of planes H . A query disk Q now corresponds to a



■ **Figure 4** (left) An illustration of the idea in \mathbb{R}^2 , the planes (here lines) of a single color, their $k_i = \left(\frac{1}{1-\alpha}\right)^i$ -levels (bright red), and the $g(\varepsilon)$ -approximate k_i -levels $L_{c,0}, L_{c,1}, \dots$ (in dark red). (right) For each i , the \mathcal{L}_i forms the lower envelope of the $L_{c,i}$ surfaces over all colors c . We search for the largest i for which q lies above \mathcal{L}_i (dashed).

vertical halfline with a top endpoint h^* , and the mode color c^* of $P \cap Q$ is the most frequently occurring color among the planes passing below h^* . Our aim is to report a color c such that at least $(1 - \varepsilon)f^*$ planes of that color pass below h^* .

The main idea to answer approximate range mode queries efficiently is to compute, for each color c , a series of $g(\varepsilon)$ -approximate k_i -levels (for some function g) considering only the planes of color c . For each choice of i , we then consider the lower envelope \mathcal{L}_i of all those k_i -levels among the various colors. See Figure 4 for an illustration. Now observe that if h^* lies in between \mathcal{L}_i and \mathcal{L}_{i+1} , the frequency f^* of a mode color c^* may only be a $g(\varepsilon)$ fraction larger than k_{i+1} , while the frequency of the color defining \mathcal{L}_i directly below h^* is at least k_i . So if $g(\varepsilon)$ and k_i/k_{i+1} are sufficiently small this is a $(1 - \varepsilon)$ -approximation. One additional complication is that even though our $g(\varepsilon)$ -approximate k_i -levels have fairly small complexity, their lower envelopes do not. So, we need to design a data structure that can test if h^* lies above or below \mathcal{L}_i without explicitly storing \mathcal{L}_i . We show that with near-linear space we can answer such queries in $O_\varepsilon(n^{1/2})$ time. This then leads to an $\tilde{O}_\varepsilon(n^{1/2})$ time query algorithm for answering k -NN queries.

We use the same approach to answer approximate queries under the L_∞ metric. Here, the approximate levels are constructed using the result of Kaplan et al. [13]. This leads to roughly the same complexities.

7 Concluding Remarks

We presented the first data structures for the chromatic k -NN problem with query times that depend only on the number of stored points. While we focused mostly on the two-dimensional case, our exact result extend to higher dimensions and other metrics as well (see full version). Our two-step approach essentially reduces the problem to efficiently answering range mode queries. The main open question is how to answer such queries efficiently. Since it is unlikely that we can answer such queries in $\Omega(n^{1/2})$ time (using only near-linear space), it is also particularly interesting to consider further improvements to the ε -approximate query data structures. For the Euclidean distance, finding the query range may now be the dominant factor (depending on the choice of ε). One option is to report a range that contains only approximately the k nearest neighbors of a query point. However, this further complicates the analysis. For the L_∞ distance it may also be possible to reduce the space usage by using a different method for computing the approximate levels (e.g. using the results by Agarwal et al. [1] or the recent results of Liu [16]).

References

- 1 Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29:912–953, 1999.
- 2 Pankaj K. Agarwal, Jirí Matousek, and Micha Sharir. On range searching with semialgebraic sets. II. *SIAM J. Comput.*, 42(6):2039–2062, 2013. doi:10.1137/120890855.
- 3 Charu C Aggarwal. *Data classification: algorithms and applications*. CRC press, 2014.
- 4 Sunil Arya, David M. Mount, and Jian Xia. Tight lower bounds for halfspace range searching. *Discrete & Computational Geometry*, 47(4):711–730, 2012. doi:10.1007/s00454-012-9412-x.
- 5 Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980. doi:10.1145/358841.358850.
- 6 Cecilia Bohler, Panagiotis Cheilaris, Rolf Klein, Chih-Hung Liu, Evanthia Papadopoulou, and Maksym Zavershynskyi. On the complexity of higher order abstract voronoi diagrams. *Computational Geometry*, 48(8):539–551, 2015. doi:10.1016/j.comgeo.2015.04.008.
- 7 Prosenjit Bose, Evangelos Kranakis, Pat Morin, and Yihui Tang. Approximate range mode and range median queries. In Volker Diekert and Bruno Durand, editors, *STACS*, pages 377–388, 2005. doi:10.1007/978-3-540-31856-9_31.
- 8 Timothy M. Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55:719–741, 2014.
- 9 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)*, volume 164 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.SoCG.2020.28.
- 10 Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.
- 11 Jerome H. Friedman, Jon Louis Bentley, and Raphael A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977. doi:10.1145/355744.355745.
- 12 W. E. Henley and D. J. Hand. A k -nearest-neighbour classifier for assessing consumer credit risk. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 45(1):77–95, 1996. URL: <http://www.jstor.org/stable/2348414>.
- 13 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. *Discrete & Computational Geometry*, 64:838–904, 2020.
- 14 Yan-Nei Law and Carlo Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. In Alípio Mário Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *Knowledge Discovery in Databases: PKDD 2005*, pages 108–120, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 15 D. T. Lee. On k -nearest neighbor voronoi diagrams in the plane. *IEEE Transactions on Computing*, 31:478–487, 1982.
- 16 Chih-Hung Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2842–2859. SIAM, 2020. doi:10.1137/1.9781611975994.173.
- 17 Chih-Hung Liu, Evanthia Papadopoulou, and Der-Tsai Lee. The k -nearest-neighbor voronoi diagram revisited. *Algorithmica*, 71(2):429–449, 2015. doi:10.1007/s00453-013-9809-9.
- 18 N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.
- 19 Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, 1979. doi:10.1287/moor.4.4.414.

67:14 Chromatic k -Nearest Neighbor Queries

- 20 D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. Chromatic nearest neighbor searching: A query sensitive approach. *Computational Geometry*, 17:97–119, 2000.
- 21 Neil Sarnak and Robert E Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986.
- 22 Robert Endre Tarjan. *Data structures and network algorithms*. SIAM, 1983.
- 23 Dan E. Willard. New data structures for orthogonal range queries. *SIAM Journal on Computing*, 14(1):232–253, 1985. doi:10.1137/0214019.

Maximum Weight b -Matchings in Random-Order Streams

Chien-Chung Huang ✉

CNRS, DI ENS, École normale supérieure, Université PSL, Paris, France

François Sellier ✉

Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

MINES ParisTech, Université PSL, F-75006, Paris, France

Abstract

We consider the maximum weight b -matching problem in the random-order semi-streaming model. Assuming all weights are small integers drawn from $[1, W]$, we present a $2 - \frac{1}{2W} + \varepsilon$ approximation algorithm, using a memory of $O(\max(|M_G|, n) \cdot \text{poly}(\log(m), W, 1/\varepsilon))$, where $|M_G|$ denotes the cardinality of the optimal matching. Our result generalizes that of Bernstein [3], which achieves a $3/2 + \varepsilon$ approximation for the maximum cardinality simple matching. When W is small, our result also improves upon that of Gamlath *et al.* [11], which obtains a $2 - \delta$ approximation (for some small constant $\delta \sim 10^{-17}$) for the maximum weight simple matching. In particular, for the weighted b -matching problem, ours is the first result beating the approximation ratio of 2. Our technique hinges on a generalized weighted version of edge-degree constrained subgraphs, originally developed by Bernstein and Stein [5]. Such a subgraph has bounded vertex degree (hence uses only a small number of edges), and can be easily computed. The fact that it contains a $2 - \frac{1}{2W} + \varepsilon$ approximation of the maximum weight matching is proved using the classical König-Egerváry's duality theorem.

2012 ACM Subject Classification Theory of computation \rightarrow Streaming, sublinear and near linear time algorithms; Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases Maximum weight matching, b -matching, streaming, random order

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.68

Related Version *Full Version:* <https://arxiv.org/abs/2207.03863>

Funding This work was funded by the grants ANR-19-CE48-0016 and ANR-18-CE40-0025-01 from the French National Research Agency (ANR).

Acknowledgements The authors thank Claire Mathieu and the anonymous reviewers for their helpful comments.

1 Introduction

The maximum weight (b -)matching problem is a classical problem in combinatorial optimization. In this paper we will study a sparsifier for that problem and use it in order to design a streaming algorithm for randomly-ordered streams of edges.

Our main tool is a generalized weighted version of the *edge-degree constrained subgraph* (EDCS), a graph sparsifier originally designed for the maximum matching problem by Bernstein and Stein [5]. Let us first recall the definition an EDCS H of a graph G [5].

► **Definition 1** (from [5]). *Let $G = (V, E)$ be a graph, and H a subgraph of G . Given any integer parameters $\beta \geq 2$ and $\beta^- \leq \beta - 1$, we say that H is a (β, β^-) -EDCS of G if H satisfies the following properties:*

- (i) For any edge $(u, v) \in H$, $\deg_H(u) + \deg_H(v) \leq \beta$
- (ii) For any edge $(u, v) \in G \setminus H$, $\deg_H(u) + \deg_H(v) \geq \beta^-$.



© Chien-Chung Huang and François Sellier;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 68; pp. 68:1–68:14

Leibniz International Proceedings in Informatics



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

An EDCS has a size that can be easily controlled by the parameter β and it somehow “balances” the vertex degrees in the graph. A very nice property of this sparsifier is that, for well-chosen values of β and β^- , it always contains a $3/2 + \varepsilon$ approximation of the maximum cardinality matching [2, 6]:

► **Theorem 2** (from the recent work of Assadi and Bernstein [2]). *Let $0 < \varepsilon < 1/2$. Set $\lambda = \frac{\varepsilon}{32}$. Let $\beta \geq \beta^- + 1$ be integers such that $\beta \geq 8\lambda^{-2} \log(1/\lambda)$ and $\beta^- \geq (1 - \lambda) \cdot \beta$. Then any (β, β^-) -EDCS H of a graph G contains a matching M_H such that $(\frac{3}{2} + \varepsilon) \cdot |M_H| \geq |M_G|$ where M_G denotes the maximum cardinality matching.*

In our paper, we generalize the EDCS in two ways:

- we handle (small) integer-weighted edges;
- we handle the more general case of b -matchings.

To describe our generalization, first let us introduce some notation. A weighted multi-graph $G = (V, E)$ is defined by its set of vertices V and its multi-set of weighted edges E drawn from $V \times V \times \{1, 2, \dots, W\}$ (i.e., $e = (u, v, k)$ represents an edge between u and v of weight $w(e) = k$). We emphasize that E is a multi-set: not only can there be multiple edges between two vertices, but also some of these edges can have the same weight. We assume that the multi-graph does not contain any self-loop. For a given vertex $v \in V$ and a given subgraph H of G , $\delta_H(v)$ denotes the multi-set of incident edges to v in H , $\deg_H(v)$ the degree of v in the multi-graph H and $\mathbf{wdeg}_H(v)$ its *weighted degree* $\sum_{(u,v,w) \in \delta_H(v)} w$ in H . Given a weighted multi-graph $G = (V, E)$ and a set of capacities $b_v \in \mathbb{Z}_+$ associated to each vertex $v \in V$, a multi-set of weighted edges M is called a b -matching if for all $v \in V$ the number of edges incident to v in M is smaller than or equal to b_v . For a given subgraph H of G , we denote by M_H an arbitrary maximum weight b -matching included in H . The concept of b -matching encompasses that of matching and allows us to tackle a larger variety of real situations where the vertices have different capacities, e.g. [20]. In this paper we will assume that the number of edges between any two vertices u and v is at most $\min(b_u, b_v)$.¹

► **Definition 3.** *Let $G = (V, E)$ be a weighted multi-graph, where E is a multiset of edges drawn from $V \times V \times \{1, 2, \dots, W\}$, $\{b_v\}_{v \in V}$ be a set of constraints, and H be a subgraph of G . Given any integer parameters $\beta \geq 3$ and $\beta^- \leq \beta - 2$, we say that H is a (β, β^-) - w - b -EDCS of G if H satisfies the following properties:*

- (i) For any edge $(u, v, w_{uv}) \in H$, $\frac{\mathbf{wdeg}_H(u)}{b_u} + \frac{\mathbf{wdeg}_H(v)}{b_v} \leq \beta \cdot w_{uv}$
- (ii) For any edge $(u, v, w_{uv}) \in G \setminus H$, $\frac{\mathbf{wdeg}_H(u)}{b_u} + \frac{\mathbf{wdeg}_H(v)}{b_v} \geq \beta^- \cdot w_{uv}$.

An EDCS is a special case of a *weighted b -EDCS* when all the b_v s and all the weights are equal to 1. We can show that such w - b -EDCSes as described in Definition 3 always exist. Moreover, we can also prove that it uses only a reasonable number of edges (up to $2\beta \cdot |M_G|$) and it contains a relatively large weighted b -matching:

► **Theorem 4.** *Let $0 < \varepsilon < 1/2$ and let W be an integer parameter. Set $\lambda = \frac{\varepsilon}{100W}$. Let $\beta \geq \beta^- + 2$ be integers such that $\frac{\beta + 6W}{\log(\beta + 6W)} \geq 2W^2 \lambda^{-2}$ and $\beta^- - 6W \geq (1 - \lambda) \cdot (\beta + 6W)$. Then any (β, β^-) - w - b -EDCS H of a weighted multi-graph G with integer edge weights bounded by W contains a b -matching M_H such that $(2 - \frac{1}{2W} + \varepsilon) \cdot w(M_H) \geq w(M_G)$.*

¹ This is actually a reasonable assumption as the maximum number of edges that are relevant between two given vertices u and v to construct a b -matching is at most $\min(b_u, b_v)$. This assumption is more debatable in the streaming setting, and this is why we explain how to handle this case in the full version of this paper.

In the full version of this paper, we give a whole class of tight examples reaching the bound of Theorem 4. Compared with the previous results of [2, 6] (Theorem 2) when $W = 1$, we can observe that the approximation ratio is the same, even though the constraints on β and β^- are a bit stricter here. Nonetheless, we can deal with b -matchings as well, even when $W > 1$. As a side note, we note that to satisfy the conditions stated in Theorem 4, it suffices that β is of order $\text{poly}(W, 1/\varepsilon)$ for some polynomial.

The *semi-streaming* model of computation [10] has been motivated by the recent rise of massive graphs, where we cannot afford to store the entire input in memory. Given that the graph is made of $|V| = n$ vertices and $|E| = m$ edges, in the semi-streaming model the graph is presented to the algorithm as a stream of edges e_1, \dots, e_m . The algorithm is allowed to make a single pass over that stream and can use a memory roughly proportional to the output size (up to a poly-logarithmic factor).

We note that in the most general model where an adversary decides the order of the elements, even for the maximum cardinality simple matching, it is still unclear whether it is possible to beat the approximation ratio of 2.

Our focus here is on the *random-order* streaming model, where the permutation of the edges in the stream is assumed to be chosen uniformly at random. This is a quite reasonable assumption as real-world data have little reason of being ordered in an adversarial way. In fact, as mentioned in [17], the random-order streaming model might better explain why certain algorithms perform better in practice than their theoretical bounds under an adversary model. It is noteworthy that under the random-order streaming model, there are already quite a few evidences to show that it is possible to beat the approximation factor of 2 [1, 3, 11, 17], at least for the simple matching.

Using an adaptation of EDCS, Bernstein [3] obtained a $3/2 + \varepsilon$ approximation in the random-order semi-streaming framework (with probability $1 - 2n^{-3}$ and using $O(n \cdot \log(n) \cdot \text{poly}(1/\varepsilon))$ memory). Similarly, we can adapt our w - b -EDCSes to design a semi-streaming algorithm for randomly-ordered streams of weighted edges:

► **Theorem 5.** *Let $0 < \varepsilon < \frac{1}{2}$ and let W be an integer parameter. There exists an algorithm that can extract with high probability (at least $1 - 2m^{-3}$) from a randomly-ordered stream of weighted edges having integer weights in $\{1, \dots, W\}$ a weighted b -matching with an approximation ratio of $2 - \frac{1}{2W} + \varepsilon$, using $O(\max(|M_G|, n) \cdot \text{poly}(\log(m), W, 1/\varepsilon))$ memory.*

Theorem 5 is the first result for the maximum (integer-weighted) b -matching problem in the random-order semi-streaming framework. For the special case of simple matching, when $W = 1$, we essentially re-capture the result of Bernstein [3] (albeit using slightly more memory). When $W > 1$, we note that prior to our work, Gamlath *et al.* [11] have obtained an approximation ratio of $2 - \delta$ for some small $\delta \sim 10^{-17}$. Our result gives a better approximation when W is reasonably small (but using a memory depending polynomially in W) and we believe that our approach is significantly simpler.

► **Remark 6.** Another generalization of EDCS has been developed by Bernstein *et al.* [4] to maintain a $3/2 + \varepsilon$ approximation of the optimal weighted matching in a dynamic graph. However it is still unknown if their construction can actually lead to an algorithm in the random-order one-pass semi-streaming model [4], or applied to b -matchings.

Technical Overview

To generalize the EDCS to the weighted case, a natural first idea is to build multiple EDCSes, one for each edge-weight from 1 to W , and then take their union. We show in the full version of this paper that such an idea does not lead to a subgraph containing a matching that is better than a 2 approximation.

Our approach is a proper generalization of EDCS, as defined in Definition 3. In Theorem 4, we show that such a w - b -EDCS contains a matching of good approximation ratio. The proof of this theorem is technically the most innovative part of the present work. In order to handle integer-weighted matchings (see Section 2) we make use of Kőnig-Egerváry's duality theorem [7] and a specially-constructed auxiliary graph. The fact that the weights of the edges are integers is critical to get an approximation ratio better than 2 (especially for Claim 13). Then, to handle b -matchings (see Section 3), we build a reduction to simple matchings and show that by doing so we do not lose too much in the approximation ratio.

Regarding Theorem 5, when we design a semi-streaming algorithm to extract a b -matching there is an additional challenge: we do not know in advance the actual size of M_G , which cannot be bounded by n (for instance $|M_G|$ could be of size $n^{1.2}$ or even larger), but we still want to use as little memory as possible, *i.e.*, $O(\max(|M_G|, n) \cdot \text{poly}(\log(m), W, 1/\varepsilon))$. We tackle this issue by using a guessing strategy in the early phase of the stream (see Section 4).

Related Work

In the adversarial semi-streaming setting, for the unweighted case, the simple greedy algorithm building a maximal matching provides a 2 approximation, which is the best known approximation ratio. Knowing whether it is possible to achieve a better approximation ratio is a major open question in the field of streaming algorithms. For weighted matchings an approximation ratio of $2 + \varepsilon$ can be achieved [12, 18, 19]. For weighted b -matchings the approximation ratio $2 + \varepsilon$ can also be attained [14]. On the hardness side, we know that an approximation ratio better than $1 + \ln 2 \approx 1.69$ cannot be achieved [15].

In contrast, for the *random-order* stream, a first result was obtained by Konrad, Magniez, and Mathieu [17] with an approximation ratio strictly below 2 for unweighted simple matchings. The approximation ratio was then improved in a sequence of papers [11, 16, 9, 3]. Currently the best result is due to Assadi and Behnezhad [1], who obtained the ratio of $3/2 - \delta$ for some small constant $\delta \sim 10^{-14}$. Regarding weighted simple matchings, Gamlath *et al.* [11] obtained an approximation ratio of $2 - \delta$ for some small constant $\delta \sim 10^{-17}$. Regarding b -matchings, to our knowledge the only result is an approximation ratio of $2 - \delta$ *in expectation* for random-order online matroid intersection by Guruganesh and Singla [13] (hence it applies for unweighted bipartite b -matchings).

2 EDCS for Weighted Matchings

In this section we consider the problem of finding a maximum weight matching in an edge-weighted graph $G = (V, E)$ where the edges have integer weights in $[1, W]$. For ease of presentation, we will use simplified notations for *simple* graphs in this section. Here $w(u, v)$ denotes edge weight between vertices u and v . For a subgraph H of G and a vertex $u \in V$, we denote by $N_H(v)$ the set of vertices adjacent to v in H , by $\deg_H(v)$ the degree of v in H , *i.e.*, $\deg_H(v) = |N_H(v)|$, and by $\mathbf{wdeg}_H(v)$ the weighted degree of v in H , *i.e.*, $\mathbf{wdeg}_H(v) = \sum_{u \in N_H(v)} w(u, v)$. For a subgraph H of G , we will denote by M_H an arbitrary maximum weight matching in H . Then we define the notion of edge-degree constrained subgraphs for weighted graphs (w -EDCS), which in fact is just Definition 3 specialized to the setting in this section.

► **Definition 7.** Let $G = (V, E)$ be a graph with weighted edges, and H be a subgraph of G . Given any integer parameters $\beta \geq 3$ and $\beta^- \leq \beta - 2$, we say that H is a (β, β^-) - w -EDCS of G if H satisfies the following properties:

- (i) For any edge $(u, v) \in H$, $\mathbf{wdeg}_H(u) + \mathbf{wdeg}_H(v) \leq \beta \cdot w(u, v)$
- (ii) For any edge $(u, v) \in G \setminus H$, $\mathbf{wdeg}_H(u) + \mathbf{wdeg}_H(v) \geq \beta^- \cdot w(u, v)$.

Here is a first simple proposition on (β, β^-) - w -EDCS (coming from Property (i)).

► **Proposition 8.** *Let H be a (β, β^-) - w -EDCS of a given graph G . Then, for all $v \in V$, we have $\deg_H(v) \leq \beta$.*

Proof. Let $v \in V$. If $N_H(v) = \emptyset$, the stated property is trivial. Otherwise, pick a vertex u such that $w(u, v) = \min_{u' \in N_H(v)} w(u', v)$. Then, by Property (i), $\beta \cdot w(u, v) \geq \text{wdeg}_H(v)$. Therefore, $\deg_H(v) \leq \frac{\text{wdeg}_H(v)}{w(u, v)} \leq \beta$, as any edge incident to v in H has a weight larger than or equal to $w(u, v)$. ◀

We show the existence of w -EDCSes by construction, using a local search algorithm. The following proof closely follows the argument of [2].

► **Proposition 9.** *Any graph $G = (V, E)$ with weighted edges contains a (β, β^-) - w -EDCS for any parameters $\beta \geq \beta^- + 2$. Such a (β, β^-) - w -EDCS can be found in $O(\beta^2 W^2 \cdot n)$ local search steps.*

Proof. Start with an empty subgraph H . Then try the following local improvements of H , until it is no longer possible. If there is an edge in H violating Property (i) of Definition 7, then fix that edge by removing it from H . Otherwise, if there is an edge in $G \setminus H$ violating Property (ii), then fix that edge by inserting it in H .

Observe that we give the priority to the correction of violations of Property (i), so that at each step of the algorithm all the vertices have degrees bounded by $\beta + 1$ (as after inserting an edge, Proposition 8 may be violated). To prove that this algorithm terminates in finite time and to show the existence of a w -EDCS, we introduce a potential function:

$$\Phi(H) = (2\beta - 2) \sum_{(u,v) \in H} w(u, v)^2 - \sum_{u \in V} (\text{wdeg}_H(u))^2.$$

As the vertices have degrees bounded by $\beta + 1$ and the edges have weights bounded by W , the value of that potential function is bounded by $2\beta^2 W^2 \cdot n$. Then we can show that after each local improvement step, the value of $\Phi(H)$ increases at least by 2 (see the full version of this paper for details). Therefore, the algorithm terminates in $O(\beta^2 W^2 \cdot n)$ steps. ◀

We also introduce the notion of w -vertex-cover of the edge-weighted graph.

► **Definition 10.** *We say that the non-negative integer variables $(\alpha_v)_{v \in V}$ represent a w -vertex-cover of a subgraph H of G if for all $(u, v) \in H$, we have $w(u, v) \leq \alpha_u + \alpha_v$. The sum $\sum_{v \in V} \alpha_v$ is called the weight of the w -vertex-cover.*

To use this data structure for the maximum weight problem, we will use the theorem of König-Egerváry [7], which is a classic duality theorem.

► **Theorem 11 (König-Egerváry).** *In any edge-weighted bipartite subgraph H of G , the maximum weight of a matching equals the smallest weight of a w -vertex-cover.*

This theorem allows us to prove the following lemma, which is technically the most important part of the present work.

► **Lemma 12.** *Let $0 < \varepsilon < 1/2$ and W be an integer parameter. For $\beta \geq \beta^- + 2$ integers such that $\frac{\beta}{\beta^-} \leq 1 + \frac{\varepsilon}{5W}$ and $\beta^- \geq \frac{4W}{\varepsilon}$, we have that any (β, β^-) - w -EDCS H of a bipartite graph G (with integer edge weights bounded by W) contains a matching M_H such that $(2 - \frac{1}{2W} + \varepsilon) \cdot w(M_H) \geq w(M_G)$.*

Proof. Using Kőnig-Egerváry's theorem in the bipartite graph H , we know that there exist integers $(\alpha_v)_{v \in V}$ such that:

- $\sum_{v \in V} \alpha_v = w(M_H)$
- for all $(u, v) \in H$, $w(u, v) \leq \alpha_u + \alpha_v$

Now consider the optimal matching M_G in G . The first idea is to use the duality theorem to relate $w(M_G)$ to $w(M_H)$, with a leftover term that will be analyzed in the second part of the proof. We introduce the notion of *good* and *bad* edges:

- the edges $(u, v) \in M_G$ such that $\beta^- \cdot w(u, v) \leq \beta \cdot (\alpha_u + \alpha_v)$, which are called *good edges*; the set of good edges is denoted as M_{good} ;
- the edges $(u, v) \in M_G$ such that $\beta^- \cdot w(u, v) > \beta \cdot (\alpha_u + \alpha_v)$, which are called *bad edges*; the set of bad edges is denoted as M_{bad} .

A key observation is that the edges in $M_G \cap H$ are necessarily good edges by the definition of the w -vertex-cover $(\alpha_v)_{v \in V}$ and the fact that $\beta^- < \beta$. Therefore, the bad edges (u, v) are in $G \setminus H$ and as a consequence they satisfy Property (ii) of Definition 7, *i.e.*, $\beta^- \cdot w(u, v) \leq \mathbf{wdeg}_H(u) + \mathbf{wdeg}_H(v)$.

Hence we can write the following:

$$\begin{aligned}
\beta^- \cdot w(M_G) &= \sum_{(u,v) \in M_{good}} \beta^- \cdot w(u, v) + \sum_{(u,v) \in M_{bad}} \beta^- \cdot w(u, v) \\
&\leq \sum_{(u,v) \in M_{good}} \beta \cdot (\alpha_u + \alpha_v) + \sum_{(u,v) \in M_{bad}} (\mathbf{wdeg}_H(u) + \mathbf{wdeg}_H(v)) \\
&= \sum_{(u,v) \in M_G} \beta \cdot (\alpha_u + \alpha_v) + \sum_{(u,v) \in M_{bad}} (\mathbf{wdeg}_H(u) + \mathbf{wdeg}_H(v) - \beta \cdot (\alpha_u + \alpha_v)) \\
&\leq \beta \cdot w(M_H) + \sum_{(u,v) \in M_{bad}} ((\mathbf{wdeg}_H(u) - \beta \cdot \alpha_u)_+ + (\mathbf{wdeg}_H(v) - \beta \cdot \alpha_v)_+),
\end{aligned}$$

where $(x)_+$ denotes the non-negative part $\max(x, 0)$. In the last inequality we also used the fact that $\sum_{(u,v) \in M_G} (\alpha_u + \alpha_v) \leq \sum_{v \in V} \alpha_v = w(M_H)$, as each vertex of V is counted at most once in that sum. Now, denoting by V_{bad} the set of vertices which are the endpoints of a bad edge and such that $\mathbf{wdeg}_H(u) - \beta \cdot \alpha_u > 0$, we get

$$\beta^- \cdot w(M_G) \leq \beta \cdot w(M_H) + \sum_{v \in V_{bad}} (\mathbf{wdeg}_H(v) - \beta \cdot \alpha_v). \quad (1)$$

Naturally, we want to upper-bound the value of $\sum_{v \in V_{bad}} (\mathbf{wdeg}_H(v) - \beta \cdot \alpha_v)$ and we will do so *via* a specially-constructed graph. Before we describe this graph, we can first easily observe that for any $v \in V_{bad}$, for any $u \in N_H(v)$, we have $w(u, v) \geq \frac{\mathbf{wdeg}_H(v)}{\beta} > \alpha_v$ (by Property (i) of Definition 7 and the definition of V_{bad}); moreover, as $(\alpha_v)_{v \in V}$ is a w -vertex-cover of H , we obtain that $\alpha_u > 0$. These observations will be useful in the following.

The new graph is $H_{bad} = (V_{bad} \cup \tilde{V}, \tilde{E})$. The vertices in H_{bad} are the vertices of V_{bad} as well as copies of the vertices of V such that $\alpha_v > 0$, *i.e.*, $\tilde{V} = \{\tilde{v} : v \in V, \alpha_v > 0\}$. We build the set of edges \tilde{E} as follows. For each $v \in V_{bad}$, for each $u \in N_H(v)$, we create in \tilde{E} an edge (v, \tilde{u}) such that $w(v, \tilde{u}) = w(v, u) - \alpha_v$ (note that if u is also in V_{bad} , then \tilde{E} will also contain another edge (u, \tilde{v}) such that $w(u, \tilde{v}) = w(u, v) - \alpha_u$). Note that $w(v, \tilde{u}) \in \mathbb{Z}_{>0}$, since $w(u, v) > \alpha_v$ as observed above. Therefore the graph H_{bad} still has non-negative integer-valued edge weights. We next remove some edges from \tilde{E} : while there exists a vertex $v \in V_{bad}$ such that $\mathbf{wdeg}_{H_{bad}}(v) > \mathbf{wdeg}_H(v) - \beta \cdot \alpha_v + W$, we pick an arbitrary edge $(v, \tilde{u}) \in \tilde{E}$ incident to v and remove it from H_{bad} . This process guarantees the following property:

$$\forall v \in V_{bad}, \mathbf{wdeg}_H(v) - \beta \cdot \alpha_v \leq \mathbf{wdeg}_{H_{bad}}(v) \leq \mathbf{wdeg}_H(v) - \beta \cdot \alpha_v + W. \quad (2)$$

This finishes the description of the graph H_{bad} . By (2), for any $(v, \tilde{u}) \in \tilde{E}$ we have:

$$\begin{aligned} \beta \cdot w(v, \tilde{u}) + W &= \beta \cdot (w(v, u) - \alpha_v) + W \geq \mathbf{wdeg}_H(v) - \beta \cdot \alpha_v + W + \mathbf{wdeg}_H(u) \\ &\geq \mathbf{wdeg}_{H_{bad}}(v) + \mathbf{wdeg}_{H_{bad}}(\tilde{u}). \end{aligned}$$

Summing this inequality over all the edges in \tilde{E} we obtain:

$$\begin{aligned} \beta \cdot w(\tilde{E}) + W \cdot |\tilde{E}| &\geq \sum_{(v, \tilde{u}) \in \tilde{E}} (\mathbf{wdeg}_{H_{bad}}(v) + \mathbf{wdeg}_{H_{bad}}(\tilde{u})) \\ &= \sum_{v \in V_{bad}} \deg_{H_{bad}}(v) \cdot \mathbf{wdeg}_{H_{bad}}(v) + \sum_{\tilde{u} \in \tilde{V}} \deg_{H_{bad}}(\tilde{u}) \cdot \mathbf{wdeg}_{H_{bad}}(\tilde{u}) \\ &\geq \sum_{v \in V_{bad}} \frac{\mathbf{wdeg}_{H_{bad}}(v)}{W} \cdot \mathbf{wdeg}_{H_{bad}}(v) + \sum_{\tilde{u} \in \tilde{V}} \frac{\mathbf{wdeg}_{H_{bad}}(\tilde{u})}{\alpha_u} \cdot \mathbf{wdeg}_{H_{bad}}(\tilde{u}) \\ &= \sum_{v \in V_{bad}} \frac{(\mathbf{wdeg}_{H_{bad}}(v))^2}{W} + \sum_{\tilde{u} \in \tilde{V}} \frac{(\mathbf{wdeg}_{H_{bad}}(\tilde{u}))^2}{\alpha_u} \\ &\geq \sum_{v \in V_{bad}} \frac{1}{W} \cdot \left(\frac{w(\tilde{E})}{|V_{bad}|} \right)^2 + \sum_{\tilde{u} \in \tilde{V}} \frac{1}{\alpha_u} \cdot \left(\frac{w(\tilde{E}) \cdot \alpha_u}{\sum_{\tilde{u}' \in \tilde{V}} \alpha_{u'}} \right)^2 \\ &= \frac{w(\tilde{E})^2}{W \cdot |V_{bad}|} + \frac{w(\tilde{E})^2}{\sum_{\tilde{u}' \in \tilde{V}} \alpha_{u'}}. \end{aligned}$$

The second inequality comes from the fact that the degree of a vertex can be lower-bounded by the weighted degree of that vertex divided by the weight of the largest edge incident to it (for $v \in V_{bad}$ this weight is W , and for $\tilde{u} \in \tilde{V}$ it is α_u , as $w(v, \tilde{u}) = w(v, u) - \alpha_v \leq \alpha_u$ for v adjacent to \tilde{u} in H_{bad}). The third inequality comes from the minimization of the function over the constraints $\sum_{v \in V_{bad}} \mathbf{wdeg}_{H_{bad}}(v) = \sum_{\tilde{u} \in \tilde{V}} \mathbf{wdeg}_{H_{bad}}(\tilde{u}) = w(\tilde{E})$. Now observing that $|\tilde{E}| \leq w(\tilde{E})$, we derive the following:

$$\beta + W \geq \frac{w(\tilde{E})}{W \cdot |V_{bad}|} + \frac{w(\tilde{E})}{\sum_{\tilde{u} \in \tilde{V}} \alpha_u}. \quad (3)$$

The following claim will help us lower bound the average weighted degree of the vertices of V_{bad} in H_{bad} , namely, $w(\tilde{E})/|V_{bad}|$. For this part it is crucial that the weights are integers.

▷ **Claim 13.** For all $(u, v) \in M_{bad}$, $(\mathbf{wdeg}_H(u) - \beta \cdot \alpha_u)_+ + (\mathbf{wdeg}_H(v) - \beta \cdot \alpha_v)_+ \geq \frac{\beta^-}{1+\varepsilon/4}$

Proof. We proceed by contradiction. Suppose that there exists $(u, v) \in M_{bad}$ such that $(\mathbf{wdeg}_H(u) - \beta \cdot \alpha_u)_+ + (\mathbf{wdeg}_H(v) - \beta \cdot \alpha_v)_+ < \frac{\beta^-}{1+\varepsilon/4}$. Then, as $\beta^- \cdot w(u, v) \leq \beta \cdot (\alpha_u + \alpha_v) + (\mathbf{wdeg}_H(u) - \beta \cdot \alpha_u)_+ + (\mathbf{wdeg}_H(v) - \beta \cdot \alpha_v)_+$, it means that

$$\beta \cdot (\alpha_u + \alpha_v) < \beta^- \cdot w(u, v) < \beta \cdot (\alpha_u + \alpha_v) + \frac{\beta^-}{1+\varepsilon/4},$$

and therefore by dividing by β^- we obtain

$$\frac{\beta}{\beta^-} \cdot (\alpha_u + \alpha_v) < w(u, v) < \frac{\beta}{\beta^-} \cdot (\alpha_u + \alpha_v) + \frac{1}{1+\varepsilon/4}.$$

As $(\alpha_u + \alpha_v) \in \{0, 1, \dots, W\}$ (recall that (u, v) is a bad edge) and because $\frac{\beta}{\beta^-} \leq 1 + \frac{\varepsilon}{5W} < 1 + \frac{\varepsilon}{4W \cdot (1 + \varepsilon/4)}$, there cannot be any integer in the open interval

$$\left] \frac{\beta}{\beta^-} \cdot (\alpha_u + \alpha_v), \frac{\beta}{\beta^-} \cdot (\alpha_u + \alpha_v) + \frac{1}{1 + \varepsilon/4} \right[,$$

implying that $w(u, v)$, which is an integer, cannot exist. The proof follows. \triangleleft

Recall that u of $(u, v) \in M_{bad}$ is part of V_{bad} only if $\mathbf{wdeg}_H(u) - \beta \cdot \alpha_u > 0$. Claim 13 then implies that given $(u, v) \in M_{bad}$, if both u and v are in V_{bad} , then $\mathbf{wdeg}_{H_{bad}}(u) + \mathbf{wdeg}_{H_{bad}}(v) \geq \frac{\beta^-}{1 + \varepsilon/4}$; if only u is in V_{bad} , then $\mathbf{wdeg}_{H_{bad}}(u) \geq \frac{\beta^-}{1 + \varepsilon/4}$. We can thus infer that $\frac{w(\tilde{E})}{|V_{bad}|} \geq \frac{\beta^-}{2 \cdot (1 + \varepsilon/4)}$ and we can rewrite (3) as $\beta + W \geq \frac{\beta^-}{2W \cdot (1 + \varepsilon/4)} + \sum_{\tilde{u} \in \tilde{V}} \frac{w(\tilde{E})}{\alpha_u}$, and therefore

$$\left(\beta + W - \frac{\beta^-}{2W \cdot (1 + \varepsilon/4)} \right) \cdot \sum_{\tilde{u} \in \tilde{V}} \alpha_u \geq w(\tilde{E}). \quad (4)$$

We now can rebound the expression of (1) as follows:

$$\begin{aligned} \beta^- \cdot w(M_G) &\leq \beta \cdot w(M_H) + \sum_{v \in V_{bad}} (\mathbf{wdeg}_H(v) - \beta \cdot \alpha_v) \\ &\leq \beta \cdot w(M_H) + \sum_{v \in V_{bad}} \mathbf{wdeg}_{H_{bad}}(v) && \text{by (2)} \\ &\leq \beta \cdot w(M_H) + w(\tilde{E}) \\ &\leq \left(2\beta + W - \frac{\beta^-}{2W \cdot (1 + \varepsilon/4)} \right) \cdot w(M_H). && \text{by (4) and } \sum_{\tilde{u} \in \tilde{V}} \alpha_u \leq w(M_H) \end{aligned}$$

Re-arranging,

$$\left(2 \frac{\beta}{\beta^-} + \frac{W}{\beta^-} - \frac{1}{2W \cdot (1 + \varepsilon/4)} \right) \cdot w(M_H) \geq w(M_G).$$

As $\frac{\beta}{\beta^-} \leq 1 + \varepsilon/4$ and $\beta^- \geq \frac{4W}{\varepsilon}$ we obtain the desired result. \triangleleft

Then we can generalize this result to non-bipartite graphs.

► **Theorem 14.** *Let $0 < \varepsilon < 1/2$ and W be an integer parameter. Set $\lambda = \frac{\varepsilon}{100W}$. For $\beta \geq \beta^- + 2$ integers such that $\frac{\beta}{\log(\beta)} \geq 2W^2 \lambda^{-2}$ and $\beta^- \geq (1 - \lambda) \cdot \beta$, we have that any (β, β^-) - w -EDCS H of a graph G (with integer edge weights bounded by W) contains a matching M_H such that $(2 - \frac{1}{2W} + \varepsilon) \cdot w(M_H) \geq w(M_G)$.*

Proof. The proof of this theorem relies on Lemma 12 and on the same construction as the one in [2], using the probabilistic method and Lovasz Local Lemma [8]. We provide the details of the proof in the full version of this paper. \triangleleft

3 EDCS for Weighted b -Matchings

From now on we consider the problem of finding a maximum weight b -matching in an edge-weighted multi-graph $G = (V, E)$. Hence we will use the notations described in the introduction. Here we recall the generalization of edge-degree constrained subgraphs (EDCS) to an edge-weighted multi-graph $G = (V, E)$ in the context of the b -matching problem.

► **Definition 15.** Let $G = (V, E)$ be a weighted multi-graph, where E is a multiset of edges drawn from $V \times V \times \{1, 2, \dots, W\}$, $\{b_v\}_{v \in V}$ be a set of constraints, and H be a subgraph of G . Given any integer parameters $\beta \geq 3$ and $\beta^- \leq \beta - 2$, we say that H is a (β, β^-) - w - b -EDCS of G if H satisfies the following properties:

- (i) For any edge $(u, v, w_{uv}) \in H$, $\frac{\text{wdeg}_H(u)}{b_u} + \frac{\text{wdeg}_H(v)}{b_v} \leq \beta \cdot w_{uv}$
- (ii) For any edge $(u, v, w_{uv}) \in G \setminus H$, $\frac{\text{wdeg}_H(u)}{b_u} + \frac{\text{wdeg}_H(v)}{b_v} \geq \beta^- \cdot w_{uv}$.

As for a w -EDCS (Proposition 8), we can bound the degree of a vertex in a w - b -EDCS H (with almost the same proof as that of Proposition 8).

► **Proposition 16.** Let H be a (β, β^-) - w - b -EDCS of a given graph G . Then, for all $v \in V$, we have $\deg_H(v) \leq \beta \cdot b_v$.

► **Proposition 17.** Let H be a (β, β^-) - w - b -EDCS of a given graph G . Then H contains at most $2\beta \cdot |M_G|$ edges.

Proof. A vertex $v \in V$ is called *saturated* by M_G if $|\delta_G(v) \cap M_G| = b_v$. We denote by V_{sat} the set of vertices saturated by M_G . As M_G is a maximal matching in G , it means that for all $(u, v, w_{uv}) \in G \setminus M_G$, either u or v is in V_{sat} . We denote by $M_{sat} \subseteq M_G$ the subset of edges in M_G that are incident to a vertex of V_{sat} . By this definition, we get:

$$\begin{aligned} |H| &= |H \cap (M_G \setminus M_{sat})| + |H \setminus (M_G \setminus M_{sat})| \leq |M_G| - |M_{sat}| + \sum_{v \in V_{sat}} \deg_H(v) \\ &\leq |M_G| - |M_{sat}| + \sum_{v \in V_{sat}} \beta \cdot b_v \leq |M_G| - |M_{sat}| + 2 \cdot |M_{sat}| \cdot \beta \leq 2\beta \cdot |M_G|, \end{aligned}$$

as for all $v \in V$, $\deg_H(v) \leq \beta \cdot b_v$ and $\sum_{v \in V_{sat}} b_v \leq 2 \cdot |M_{sat}|$. ◀

We can also show that such w - b -EDCSes always exist.

► **Proposition 18.** Any multi-graph $G = (V, E)$, along with a set of constraints $\{b_v\}_{v \in V}$, contains a (β, β^-) - w - b -EDCS for any parameters $\beta \geq \beta^- + 2$. Such a (β, β^-) - w - b -EDCS can also be found in $O(\beta^2 W^2 \cdot |M_G|)$ local search steps.

Proof. As in the proof of Proposition 9, we follow closely the argument of [2]. We use the same local-search algorithm as the one in Proposition 9, except that the properties violated are those of Definition 15. Here we also give the priority to the correction of violations of Property (i), so that the at each step of the algorithm all the vertices $v \in V$ have degrees bounded by $\beta \cdot b_v + 1$. To prove that this algorithm terminates and show the existence of a w - b -EDCS, we introduce the following potential function:

$$\Phi(H) = (2\beta - 2) \sum_{(u, v, w_{uv}) \in H} w_{uv}^2 - \sum_{u \in V} \frac{(\text{wdeg}_H(u))^2}{b_u}.$$

Observe that because of Proposition 17, the value of $\Phi(H)$ is bounded by $2\beta W^2 \cdot 2\beta \cdot |M_G|$. We can also show that after each local improvement, the value of $\Phi(H)$ increases by at least $3/2$ (see full version for details). Hence the algorithm terminates in $O(\beta^2 W^2 \cdot |M_G|)$ steps. ◀

The main interest of these w - b -EDCSes is that they contain an (almost) $2 - \frac{1}{2W}$ approximation, as in the case of w -EDCSes in simple graphs (Theorem 14).

► **Theorem 4.** *Let $0 < \varepsilon < 1/2$ and let W be an integer parameter. Set $\lambda = \frac{\varepsilon}{100W}$. Let $\beta \geq \beta^- + 2$ be integers such that $\frac{\beta+6W}{\log(\beta+6W)} \geq 2W^2\lambda^{-2}$ and $\beta^- - 6W \geq (1 - \lambda) \cdot (\beta + 6W)$. Then any (β, β^-) - w - b -EDCS H of a weighted multi-graph G with integer edge weights bounded by W contains a b -matching M_H such that $(2 - \frac{1}{2W} + \varepsilon) \cdot w(M_H) \geq w(M_G)$.*

Proof. Consider a maximum weight b -matching M_G . We will build from H and M_G two simple graphs $G' = (V', E')$ and $H' = (V', E'_H)$. The set of vertices V' contains, for each vertex $v \in V$, b_v vertices v_1, \dots, v_{b_v} , so that V' contains $\sum_{v \in V} b_v$ vertices in total. To construct E' , for each $v \in V$, we will distribute the edges of $\delta(v) \cap (H \cup M_G)$ among the b_v vertices v_1, \dots, v_{b_v} in such a way so that the following three properties hold:

- (i) each v_i has a most one edge of M_G incident to it;
- (ii) G' is a simple graph;
- (iii) each v_i has a weighted degree in the interval $\left[\frac{\mathbf{wdeg}_H(v)}{b_v} - 2W, \frac{\mathbf{wdeg}_H(v)}{b_v} + 3W \right]$.

The existence of such a distribution is achieved by a greedy procedure (see the full version of this paper for a proof of this fact). For Property (ii), it is crucial that the graph G has at most $\min(b_u, b_v)$ edges between any vertices u and v . This property is important in the proof of Theorem 14 (where negative association is used). Then, for H' , we just consider the restriction of G' to the edges corresponding to H (ignoring those from $M_G \setminus H$ in the preceding construction).

Observe that M_G corresponds a simple matching in G' , and that any simple matching in H' corresponds to a b -matching in H . Next we show that H' is a $(\beta + 6W, \beta^- - 6W)$ -EDCS for simple graph G' . Consider an edge $(u_i, v_j) \in H'$. It corresponds to an edge (u, v, w_{uv}) of H so $\mathbf{wdeg}_{H'}(u_i) + \mathbf{wdeg}_{H'}(v_j) \leq \frac{\mathbf{wdeg}_H(u)}{b_u} + \frac{\mathbf{wdeg}_H(v)}{b_v} + 6W \leq (\beta + 6W) \cdot w_{uv}$, so Property (i) of Definition 7 holds. Consider next an edge $(u_i, v_j) \in G' \setminus H'$. It corresponds to an edge (u, v, w_{uv}) of $M_G \setminus H$, so $\mathbf{wdeg}_{H'}(u_i) + \mathbf{wdeg}_{H'}(v_j) \geq \frac{\mathbf{wdeg}_H(u)}{b_u} + \frac{\mathbf{deg}_H(v)}{b_v} - 6W \geq (\beta^- - 6W) \cdot w_{uv}$ (as there can be a difference of at most W between the weighted degree of u in G' and in H'). Thus Property (ii) of Definition 7 holds as well. To conclude, H' is a $(\beta + 6W, \beta^- - 6W)$ - w -EDCS of G' , so by Theorem 14 we have that $(2 - \frac{1}{2W} + \varepsilon) \cdot w(M_{H'}) \geq w(M_{G'}) = w(M_G)$. As $w(M_H) \geq w(M_{H'})$ (because any matching in H' corresponds to a b -matching of the same weight in H), completing the proof. ◀

4 Application to b -Matchings in Random-Order Streams

In this section we consider the random-order semi-streaming model and we show how our results in the preceding section can be adapted to get a $2 - \frac{1}{2W} + \varepsilon$ approximation.

As our algorithm builds on that of Bernstein [3] for the unweighted simple matching, let us briefly summarize his approach. In the first phase of the streaming, he constructs a subgraph that satisfies only a weaker definition of EDCS in Definition 1 (only Property (i) holds). In the second phase of the streaming, he collects the “underfull” edges, which are those edges that violate Property (ii). He shows that in the end, the union of the subgraph built in the first phrase and the underfull edges collected in the second phase, with high probability, contains a $3/2 + \varepsilon$ approximation and that the total memory used is in the order of $O(n \cdot \log n)$. As we will show below, this approach can be adapted to our context of edge-weighted b -matching. Our main technical challenge lies in the fact that unlike the simple matching, the size of M_G can vary a lot. We need a “guessing” strategy to ensure that the required memory is proportional to $|M_G|$.

► **Definition 19.** *We say that a graph H has bounded weighted edge-degree β if for every edge $(u, v, w_{uv}) \in H$, $\frac{\mathbf{wdeg}_H(u)}{b_u} + \frac{\mathbf{wdeg}_H(v)}{b_v} \leq \beta \cdot w_{uv}$.*

► **Definition 20.** Let G be a edge-weighted multi-graph, and let H be a subgraph of G with bounded weighted edge-degree β . For any parameter β^- , we say that an edge $(u, v, w_{uv}) \in G \setminus H$ is (H, β, β^-) -underfull if $\frac{wdeg_H(u)}{b_u} + \frac{wdeg_H(v)}{b_v} < \beta^- \cdot w_{uv}$.

► **Lemma 21.** Let $0 < \varepsilon < 1/2$ be any parameter and W be an integer parameter. Set $\lambda = \frac{\varepsilon}{100W}$. Suppose that β^- and $\beta \geq \beta^- + 2$ are integers so that $\frac{\beta+8W}{\log(\beta+8W)} \geq 2W^2\lambda^{-2}$ and $\beta^- - 6W \geq (1 - \lambda) \cdot (\beta + 8W)$. Given an edge-weighted multi-graph G with integer weights in $1, \dots, W$, and a subgraph H with bounded weighted edge-degree β , if X contains all edges in $G \setminus H$ that are (H, β, β^-) -underfull, then $(2 - \frac{1}{2W} + \varepsilon) \cdot w(M_{H \cup X}) \geq w(M_G)$.

Proof. First, observe that $H \cup X$ is not necessarily a w - b -EDCS of G . Thereby we use another argument from [3]. Let M_G be a maximum weight b -matching in G , let $M_G^H = M_G \cap H$ and $M_G^{G \setminus H} = M_G \cap (G \setminus H)$. Let $X^M = X \cap M_G^{G \setminus H}$. We can observe that $w(M_G) = w(M_{H \cup M_G^{G \setminus H}})$. Then we can show that $H \cup X^M$ is a $(\beta + 2W, \beta^-)$ - w - b -EDCS of $H \cup M_G^{G \setminus H}$ (see the full version of this paper). As a result, Theorem 4 can be applied in this case and we get that $(2 - \frac{1}{2W} + \varepsilon) \cdot w(M_{H \cup X^M}) \geq w(M_{H \cup M_G^{G \setminus H}}) = w(M_G)$, thus concluding the proof. ◀

► **Remark 22.** One can easily notice that there exist integers β and β^- that are $O(\text{poly}(W, 1/\varepsilon))$ satisfying the conditions of Lemma 21. From now on, we will use the parameters λ, β , and β^- satisfying the conditions of Lemma 21 and they are of values $O(\text{poly}(W, 1/\varepsilon))$.

■ **Algorithm 1** Main algorithm computing a weighted b -matching for a random-order stream.

```

1:  $H \leftarrow \emptyset$ 
2:  $\forall 0 \leq i \leq \log_2 m, \alpha_i \leftarrow \left\lfloor \frac{\varepsilon \cdot m}{\log_2(m) \cdot (2^{i+2} \beta^2 W^2 + 1)} \right\rfloor$ 
3: for  $i = 0 \dots \log_2 m$  do
4:   PROCESSSTOPPED  $\leftarrow$  FALSE
5:   for  $2^{i+2} \beta^2 W^2 + 1$  iterations do
6:     FOUNDUNDERFULL  $\leftarrow$  FALSE
7:     for  $\alpha_i$  iterations do
8:       let  $(u, v, w_{uv})$  be the next edge in the stream
9:       if  $\frac{wdeg_H(u)}{b_u} + \frac{wdeg_H(v)}{b_v} < \beta^- \cdot w_{uv}$  then
10:        add edge  $(u, v, w_{uv})$  to  $H$ 
11:        FOUNDUNDERFULL  $\leftarrow$  TRUE
12:        while there exists  $(u', v', w_{u'v'}) \in H : \frac{wdeg_H(u')}{b_{u'}} + \frac{wdeg_H(v')}{b_{v'}} > \beta \cdot w_{u'v'}$  do
13:          remove  $(u', v', w_{u'v'})$  from  $H$ 
14:        if FOUNDUNDERFULL = FALSE then
15:          PROCESSSTOPPED  $\leftarrow$  TRUE
16:          break from the loop
17:   if PROCESSSTOPPED = TRUE then
18:     break from the loop
19:  $X \leftarrow \emptyset$ 
20: for each  $(u, v, w_{uv})$  remaining edge in the stream do
21:   if  $\frac{wdeg_H(u)}{b_u} + \frac{wdeg_H(v)}{b_v} < \beta^- \cdot w_{uv}$  then
22:     add edge  $(u, v, w_{uv})$  to  $X$ 
23: return the maximum weight  $b$ -matching in  $H \cup X$ 

```

The algorithm, formally described in Algorithm 1, consists of two phases. The first phase, corresponding to Lines 3-18, constructs a subgraph H of bounded weighted edge-degree β using only a ε fraction of the stream E^{early} . In the second phase, the algorithm collects the

68:12 Maximum Weight b -Matchings in Random-Order Streams

underfull edges in the remaining part of the stream E^{late} . As in [3] we use the idea that if no underfull edge was found in an interval of size α (see Lines 6-13), with high probability the number of underfull edges remaining in the stream is bounded by some value $\gamma = 4 \log(m) \frac{m}{\alpha}$. The issue is therefore to choose the right size of interval α , because we do not know the order of magnitude of $|M_G|$ in the b -matching problem: if we do as in [3] by choosing only one fixed size of intervals α , then if α is too small, the value of γ will be too big compared to $|M_G|$, whereas if the value of α is too large we will not be able to terminate the first phase of the algorithm within the early fraction of size εm . Therefore, the idea in the first phase of the algorithm is to “guess” the value of $\log_2 |M_G|$ by trying successively larger and larger values of i (see Line 3). In fact, by setting $i_0 = \lceil \log_2 |M_G| \rceil$, we know that the number of operations that can be performed on a w - b -EDCS is bounded by $2^{i_0+2} \beta^2 W^2$ (see the proof of Proposition 18). As a result we know that the first phase should always stop at a time where i is smaller than or equal to i_0 , and therefore at a time when $\alpha_i \geq \alpha_{i_0}$. Then we can prove that with high probability the number of remaining underfull edges in the stream is at most $\gamma_i = 4 \log(m) \frac{m}{\alpha_i}$.

Algorithm 1 works when M_G is neither too small nor too big. Here we will first argue that the other border cases can be handled anyway. We first have this easy lemma (its proof is very similar to that of Proposition 17, see the full version of this paper).

► **Lemma 23.** *We have the inequality $|G| \leq 2n \cdot |M_G|$.*

Then we use it to handle the case of small b -matchings.

▷ **Claim 24.** We can assume that $w(M_G) \geq \frac{3W^2}{2\varepsilon^2} \log(m)$.

Proof. In fact, if $w(M_G) < \frac{3W^2}{2\varepsilon^2} \log(m)$, then $|M_G| < \frac{3W^2}{2\varepsilon^2} \log(m)$ and by Lemma 23 the graph has only $m = O(n \cdot \frac{3W^2}{2\varepsilon^2} \cdot \log(m))$ edges, so the whole graph can be stored only using $O(n \cdot \text{poly}(\log(m), W, 1/\varepsilon))$ memory, implying that we can compute an exact solution. ◁

▷ **Claim 25.** Assuming Claim 24, with probability at least $1 - m^{-3}$ the late part of the stream E^{late} contains at least a $(1 - 2\varepsilon)$ fraction of the optimal b -matching.

Proof. Consider a maximum weight b -matching $M_G = \{f_1, \dots, f_{|M_G|}\}$. We define the random variables $X_i = \mathbb{1}_{f_i \in E^{early}} \cdot w(f_i)$. Hence we have $\mathbb{E}[\sum X_i] = \varepsilon \cdot w(M_G)$. Moreover, the random variables X_i are negatively associated, so we can use Hoeffding’s inequality to get $\mathbb{P}\left[\sum_{i=1}^{|M_G|} X_i \geq 2\varepsilon \cdot w(M_G)\right] \leq \exp\left(-\frac{2 \cdot \varepsilon^2 \cdot w(M_G)^2}{|M_G| \cdot W^2}\right) \leq \exp\left(-\frac{2 \cdot \varepsilon^2 \cdot w(M_G)}{W^2}\right) \leq m^{-3}$, as we now assume that $w(M_G) \geq \frac{3W^2}{2\varepsilon^2} \log(m)$ (see Claim 24). ◁

Recall that we defined $i_0 = \lceil \log_2 |M_G| \rceil$.

▷ **Claim 26.** We can assume that $\frac{\varepsilon \cdot m}{\log_2(m) \cdot (2^{i_0+2} \beta^2 W^2 + 1)} \geq 1$.

Proof. If this is not the case, then we can just store all the edges of G as the number of edges m is bounded by $\frac{\log_2(m) \cdot (2^{i_0+2} \beta^2 W^2 + 1)}{\varepsilon} = O(|M_G| \cdot \text{poly}(\log(m), W, 1/\varepsilon))$ (as β is $O(\text{poly}(W, 1/\varepsilon))$, see Remark 22). As a result, if at some point of the first phase we have not stopped and we have $\alpha_i = 0$, then we store all the remaining edges of E^{late} and we will be able to get a $(1 - 2\varepsilon)$ approximation with high probability (because of Claim 25) using $O(|M_G| \cdot \text{poly}(\log(m), W, 1/\varepsilon))$ memory. ◁

Then we can move on to our main algorithm. The following lemma is very similar to the one used in [3] (see the proof in the full version of this paper). It can then be combined with previous lemmas and claims to prove that a $2 - \frac{1}{2W} + \varepsilon$ approximation can be achieved with high probability.

► **Lemma 27.** *The first phase of Algorithm 1 uses $O(\beta \cdot |M_G|)$ memory and constructs a subgraph H of G , satisfying the following properties:*

1. *The first phase terminates within the first εm edges of the stream.*
2. *When the first phase terminates after processing some edge, we have:*
 - a. *H has bounded weighted edge degree β , and contains at most $O(\beta \cdot |M_G|)$ edges.*
 - b. *With probability at least $1 - m^{-3}$, the total number of (H, β, β^-) -underfull edges in the remaining part of the stream is at most $\gamma = O(|M_G| \cdot (\log(m))^2 \cdot \beta^2 W^2 \cdot 1/\varepsilon)$.*

► **Theorem 28.** *Let $\varepsilon > 0$. Using Algorithm 1, with probability $1 - 2m^{-3}$, one can extract from a randomly-ordered stream of edges a weighted b -matching with an approximation ratio of $2 - \frac{1}{2W} + \varepsilon$, using $O(\max(|M_G|, n) \cdot \text{poly}(\log(m), W, 1/\varepsilon))$ memory.*

Proof. Applying Lemma 21 to the graph $H \cup G^{\text{late}}$ we can get, choosing the right values β and β^- (which are $O(\text{poly}(W, 1/\varepsilon))$), $H \cup X$ contains a $(1 - 2\varepsilon)^{-1} \cdot (2 - \frac{1}{2W} + \varepsilon)$ approximation of the optimal b -matching (with probability at least $1 - m^{-3}$, see Claim 25), and with a memory consumption of $O(|M_G| \cdot \text{poly}(\log(m), W, 1/\varepsilon))$ (with probability at least $1 - m^{-3}$, see Lemma 27), with probability at least $1 - 2m^{-3}$ (union bound). Hence the proof. ◀

References

- 1 Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. 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 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.19.
- 2 Sepehr Assadi and Aaron Bernstein. Towards a unified theory of sparsification for matching problems. 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 11:1–11:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASICS.SOSA.2019.11.
- 3 Aaron Bernstein. Improved Bounds for Matching in Random-Order Streams. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:13, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.ICALP.2020.12.
- 4 Aaron Bernstein, Aditi Dudeja, and Zachary Langley. A framework for dynamic matching in weighted graphs. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 668–681. ACM, 2021. doi:10.1145/3406325.3451113.
- 5 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. 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 167–179. Springer, 2015. doi:10.1007/978-3-662-47672-7_14.
- 6 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. 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 692–711. SIAM, 2016. doi:10.1137/1.9781611974331.ch50.
- 7 Jenő Egerváry. Matrixok kombinatorikus tulajdonságairól. *Matematikai és Fizikai Lapok*, 38(1931):16–28, 1931.
- 8 Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Colloquia Mathematica Societatis Janos Bolyai 10. Infinite and Finite Sets, Keszthely (Hungary)*. Citeseer, 1973.

- 9 Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. 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 1773–1785. SIAM, 2020. doi:10.1137/1.9781611975994.108.
- 10 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348:207–216, December 2005. doi:10.1016/j.tcs.2005.09.013.
- 11 Buddhima Gamblath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC '19*, pages 491–500, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331603.
- 12 Mohsen Ghaffari and David Wajc. Simplified and Space-Optimal Semi-Streaming $(2+\epsilon)$ -Approximate Matching. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, volume 69 of *OpenAccess Series in Informatics (OASICs)*, pages 13:1–13:8, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASICs.SOSA.2019.13.
- 13 Guru Prashanth Guruganesh and Sahil Singla. Online matroid intersection: Beating half for random arrival. In Friedrich Eisenbrand and Jochen Könnemann, editors, *Integer Programming and Combinatorial Optimization – 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, volume 10328 of *Lecture Notes in Computer Science*, pages 241–253. Springer, 2017. doi:10.1007/978-3-319-59250-3_20.
- 14 Chien-Chung Huang and François Sellier. Semi-streaming algorithms for submodular function maximization under b -matching constraint. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*, 2021.
- 15 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 1874–1893. SIAM, 2021. doi:10.1137/1.9781611976465.112.
- 16 Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPICs*, pages 74:1–74:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.MFCS.2018.74.
- 17 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2012. doi:10.1007/978-3-642-32512-0_20.
- 18 Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. In *SODA*, pages 1914–1933, 2021.
- 19 Ami Paz and Gregory Schwartzman. A $(2 + \epsilon)$ -approximation for maximum weight matching in the semi-streaming model. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2153–2161, 2017. doi:10.1137/1.9781611974782.140.
- 20 Moshe Tennenholtz. Some tractable combinatorial auctions. In *AAAI/IAAI*, pages 98–103, 2000.

Embedding Phylogenetic Trees in Networks of Low Treewidth

Leo van Iersel 

Delft Institute of Applied Mathematics, Delft University of Technology, The Netherlands

Mark Jones  

Delft Institute of Applied Mathematics, Delft University of Technology, The Netherlands

Mathias Weller  

CNRS, LIGM (UMR 8049), Champs-s/-Marne, France

Abstract

Given a rooted, binary phylogenetic network and a rooted, binary phylogenetic tree, can the tree be embedded into the network? This problem, called TREE CONTAINMENT, arises when validating networks constructed by phylogenetic inference methods. We present the first algorithm for (rooted) TREE CONTAINMENT using the treewidth t of the input network N as parameter, showing that the problem can be solved in $2^{O(t^2)} \cdot |N|$ time and space.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases fixed-parameter tractability, treewidth, phylogenetic tree, phylogenetic network, display graph, tree containment, embedding

Digital Object Identifier 10.4230/LIPIcs.EISA.2022.69

Related Version *Full Version:* <https://arxiv.org/abs/2207.00574v2> [40]

Funding *Leo van Iersel:* Partially funded by Netherlands Organization for Scientific Research (NWO) Vidi grant 639.072.602 and KLEIN grant OCENW.KLEIN.125.

Mark Jones: Partially funded by Netherlands Organization for Scientific Research (NWO) KLEIN grant OCENW.KLEIN.125.

Acknowledgements We are extremely grateful to the anonymous reviewers for their many insightful and helpful comments.

1 Introduction

1.1 Background: phylogenetic trees and networks

Phylogenetic trees and networks are graphs used to represent evolutionary relationships. In particular, a *rooted phylogenetic network* is a directed acyclic graph with distinctly labelled leaves, a unique root and no indegree-1 outdegree-1 vertices. The labels of the leaves can, for example, represent a collection of studied biological species, and the network then describes how they evolved from a common ancestor (the root). Here, we will only consider rooted binary phylogenetic networks, which we will call *networks* for short. Vertices with indegree 2 in such a network are called *reticulations* and represent events where lineages combine, for example the emergence of new hybrid species. A network without reticulations is a *phylogenetic tree*.

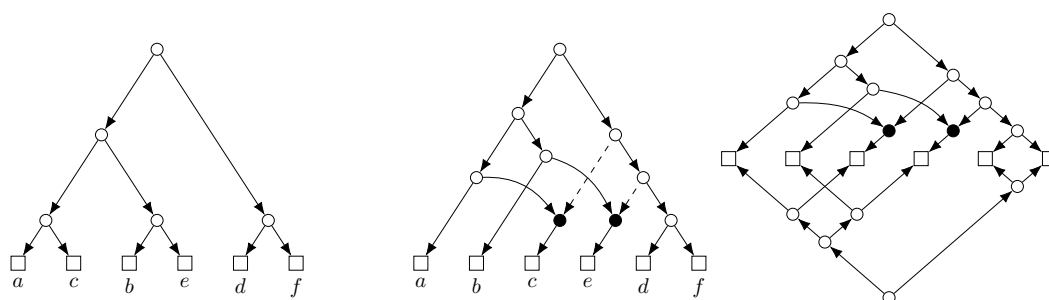


© Leo van Iersel, Mark Jones, and Mathias Weller;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 69; pp. 69:1–69:14
Leibniz International Proceedings in Informatics



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



■ **Figure 1** **Left:** a phylogenetic tree T . **Middle:** a phylogenetic network N displaying T (solid lines indicate an embedding of T ; black nodes indicate reticulations). **Right:** the display graph $D(N, T)$ of N and T (see Section 1.5) with the network part drawn on top and the tree part drawn on the bottom. Note that vertices of the display graph are not labelled. In the figure, the leaves (square vertices) are ordered in the same way as in N .

1.2 The Tree Containment problem

The evolutionary history of a small unit of hereditary information (for example a gene, a fraction of a gene or (in linguistics) a word) can often be described by a phylogenetic tree. This is because at each reticulation, each unit is inherited from only one parent. Hence, if we trace back the evolutionary history of such a hereditary unit in the network, we see that its phylogenetic tree can be embedded in the network. This raises the fundamental question: given a phylogenetic network and a phylogenetic tree, can the tree be embedded into the network? This is called the TREE CONTAINMENT problem (see Figure 1). To formalize this problem, we say that a network N *displays* a tree T if some subgraph of N is a subdivision of T .

TREE CONTAINMENT (TC)

Input: phylogenetic network N_{IN} and tree T_{IN} , both on the same set of leaf labels

Question: Does N_{IN} display T_{IN} ?

1.2.1 Motivation

Apart from being a natural and perhaps one of the most fundamental questions regarding phylogenetic networks, the TREE CONTAINMENT problem has direct applications in phylogenetics. The main application is the validation of phylogenetic network inference methods. After constructing a network, one may want to verify whether it is consistent with the phylogenetic trees. For example, if a heuristic method is used to generate a network for a genomic data set, and tree inference methods are used to generate trees for each gene, then the quality of the produced network can be assessed by computing the fraction of the gene trees that can be embedded into it. In addition, one may want to find the actual embeddings for visualisation purposes and/or to assess the importance of each network arc.

However, our main motivation for studying TREE CONTAINMENT is that it is a first step towards the wider application of treewidth based approaches in phylogenetics (see Sections 1.4 and 1.5). The techniques we develop are not exclusively designed for TREE CONTAINMENT but intended to be useful also for other problems such as NETWORK CONTAINMENT [31] and HYBRIDIZATION NUMBER [8, 41, 42, 43]. The former is the natural generalization of TREE CONTAINMENT in which we have two networks as input and want to decide whether one can be embedded into the other. It can, in particular, be used to decide whether two networks

are isomorphic. In the latter problem, HYBRIDIZATION NUMBER, the input consists of a set of phylogenetic trees, and the aim is to construct a network with at most k reticulations that embeds each of the input trees. Although this will certainly be non-trivial, we expect that at least part of the approach we introduce here can be applied to those and other problems in phylogenetics.

1.3 Previous work

TREE CONTAINMENT was shown to be NP-hard [32], even for tree-sibling, time-consistent, regular networks [29]. On the positive side, polynomial-time algorithms were found for other restricted classes, including tree-child networks [17, 18, 22, 24, 29, 44]. The first non-trivial FPT algorithm for TREE CONTAINMENT on general networks had running time $O(2^{k/2}n^2)$, where the parameter k is the number of reticulations in the network [32]. Another algorithm was proposed by [23] with the same parameter, but it is only shown to be FPT for a restricted class of networks. Since the problem can be split into independent subproblems at non-leaf cut-edges [29], the parameterization can be improved to the largest number of reticulations in any biconnected component (block), also called the *level* of the network. Further improving the parameterization, the maximum number t^* of “unstable component-roots” per biconnected component was considered and an algorithm (working also in the non-binary case) was found with running time $O(3^{t^*} |N||T|)$ [44]. Herein, a parameterization “improves” over another if the first is provably smaller than (a function of) the second in any input network.

Several generalizations and variants of the TREE CONTAINMENT problem have been studied. The more general NETWORK CONTAINMENT problem asks to embed a *network* in another and has been shown to be solvable in polynomial time on a restricted network class [31]. When allowing multifurcations and non-binary reticulations, two variants of TREE CONTAINMENT have been considered: In the FIRM version, each non-binary node (“polytomy”) of the tree has to be embedded in a polytomy of the network whereas, in the SOFT version, polytomies may be “resolved” into binary subtrees in any way [2]. Finally, the unrooted version of TREE CONTAINMENT was also shown to be NP-hard but fixed-parameter tractable when the parameter is the reticulation number (the number of edges that need to be deleted from the network to obtain a tree) [28]. While this version of the problem is also known to be fixed-parameter tractable with respect to the treewidth of the network [30], the work does not explicitly describe an algorithm and the implied running time depends on Courcelle’s theorem [10] which makes practical implementation virtually impossible.

Since the notion of “display” closely resembles that of “topological minor” (with the added constraint that the embedding must respect leaf-labels), TREE CONTAINMENT can be understood as a special case of a variant of the well-known TOPOLOGICAL MINOR CONTAINMENT (TMC) problem for directed graphs. TMC is known to be NP-complete in general by reduction from HAMILTONIAN CYCLE and previous algorithmic results focus on the undirected variant, parameterized by the *size* h of the sought topological minor H (corresponding to the input tree for TREE CONTAINMENT). In particular, undirected TMC can be solved in $f(h)n^{O(1)}$ time [21, 15]. In the directed case, even the definition of “topological minor” has been contested [19] and we are aware of little to no algorithmic results. In TREE CONTAINMENT, part of the embedding of the host tree in the guest network is fixed by the leaf-labeling. If the node-mapping is fixed for *all* nodes of the host, then directed TMC generalizes the DISJOINT PATHS problem [16], which is NP-complete for 2 paths or in case the host network is acyclic. Indeed, one can show TREE CONTAINMENT to be NP-hard in a similar fashion [32].

1.4 Treewidth

From the overview presented in Section 1.3, we see that the parameters that are most heavily used are the reticulation number and the level. This is true not only for TREE CONTAINMENT but more generally in the phylogenetic networks literature. Although these parameters are natural, their downside is that they are not necessarily much smaller than the input size. This is why we study a different parameter here.

The *treewidth* of a graph measures its tree-likeness (see definition below), similarly to the reticulation number and level. In that sense, it is also a natural parameter to consider in phylogenetics, where networks are often expected to be reasonably tree-like. A major advantage of treewidth is that it is expected to be much smaller than the reticulation number and level. In particular, there exist classes of networks for which the treewidth is at most a constant factor times the square-root of the level (see [33] for an example). Moreover, a broad range of advanced techniques have been developed for designing FPT algorithms for graph problems when the parameter is the treewidth [4, 12, 6, 13]. For these reasons, the treewidth has recently been studied for phylogenetics problems [30, 34, 33, 38] and related width parameters have been proposed [3]. However, using treewidth as parameter for phylogenetic problems poses major challenges and, therefore, there are still few algorithms in phylogenetics that use treewidth as parameter (see Section 1.5).

It will be convenient to define a *tree decomposition* of a graph $G = (V, E)$ as a rooted tree, where each vertex of the tree is called a *bag* and is assigned a partition (P, S, F) of V , where S is a separator between P and F . We will refer to S as the *present* of the bag. The set P is equal to the union of the presents of all descendant bags (minus the elements of S) and we refer to it as the *past* of the bag. The set $F = V \setminus (S \cup P)$ is referred to as the *future* of the bag. For each edge of the graph, there is at least one bag for which both endpoints of the edge are in the present of the bag. Finally, for each $v \in V$, the bags that have v in the present form a non-empty connected subtree of the tree decomposition. The *width* of a tree decomposition is one less than the maximum size of any bag's present and the *treewidth* $tw(G)$ of a graph G is the minimum width of any tree decomposition of G . The treewidth of a phylogenetic network or other directed graph is the treewidth of the underlying undirected graph.

Our dynamic programming works with *nice* tree decompositions, in which the root is assigned $(V, \emptyset, \emptyset)$ and each bag assigned (P, S, F) has exactly one of four types: *Leaf* bags have $P = S = \emptyset$ (hence $F = V$) and have no child, *Introduce* bags have a single child assigned $(P, S \setminus \{z\}, F \cup \{z\})$ for some $z \in S$, *Forget* bags have a single child assigned $(P \setminus \{z\}, S \cup \{z\}, F)$ for some $z \in P$, and *Join* bags have two children assigned $(L, S, F \cup R)$ and $(R, S, F \cup L)$ respectively, where (L, R) is a partition of P . When the treewidth is bounded by a constant, [5] showed that a minimum-width tree decomposition can be found in linear time and [36] showed that a nice tree decomposition of the same width can be obtained in linear time. Regarding approximation, it is known that, for all graphs G , tree decompositions of width $O(tw(G))$ can be computed in time single-exponential in $tw(G)$ [11, 7, 37] and tree decompositions of width $O(tw(G)\sqrt{\log tw(G)})$ can be computed in polynomial time [14].

1.5 Challenges

One of the main challenges of using treewidth as parameter in phylogenetics is that the central goal in this field is to infer phylogenetic networks and, thus, the network is not known *a priori* so a tree decomposition cannot be constructed easily. A possible strategy to overcome this problem is to work with the *display graph* (see Figure 1). Consider a problem

taking as input a set of trees, such as HYBRIDIZATION NUMBER. Then, the *display graph* of the trees is obtained by taking all trees and identifying leaves with the same label. Now we have a graph in the input and hence we can compute a tree decomposition. Moreover, in some cases, there is a strong relation between the treewidth of the display graph and the treewidth of an optimal network [20, 33, 30].

A few instances of exploiting (tree decompositions of) the display graph of input networks for algorithm design have been published. Famously, Bryant and Lagergren [9] designed MSOL formulations solving the TREE CONSISTENCY problem on display graphs, which have been improved¹ by a concrete dynamic programming on a given tree decomposition of the display graph [1]. Kelk et al. [34] also developed MSOL formulations on display graphs for multiple incongruence measures on trees, based on so-called “agreement forests”. For the TREE CONTAINMENT problem, MSOL formulations acting on the display graph have been used to prove fixed-parameter tractability with respect to the treewidth [30]. Analogously to the work of Baste et al. [1] for TREE CONSISTENCY, we develop in this manuscript a concrete dynamic programming algorithm for TREE CONTAINMENT acting on display graphs.

TREE CONTAINMENT is conceptually similar to HYBRIDIZATION NUMBER in the sense that the main challenge is to decide which tree vertices correspond to which vertices of the other trees (for HYBRIDIZATION NUMBER) or network vertices (for TREE CONTAINMENT). However, HYBRIDIZATION NUMBER is even more challenging since the network may contain vertices that do not correspond to any input vertex [41]. Therefore, TREE CONTAINMENT is a natural first problem to develop techniques for, aiming at extending them to HYBRIDIZATION NUMBER and other problems in phylogenetics in the long run.

That being said, solving TREE CONTAINMENT parameterized by treewidth poses major challenges itself. Even though the general idea of dynamic programming on a tree decomposition is clear, its concrete use for TREE CONTAINMENT is severely complicated by the fact that the tree decomposition does not know the correspondence between tree vertices and network vertices. For example, when considering a certain bag of the tree decomposition, a tree vertex that is in the present of that bag may have to be embedded into a network vertex that is in the past or in the future. It may also be necessary to map vertices from the future of the tree to the past of the network and vice versa. Therefore, it will not be possible to “forget the past” and “not worry about the future”. In particular, this makes it much more challenging to bound the number of possible assignments for a given bag. We will do this by bounding the number of “time-travelling” vertices by a function of the treewidth. We will describe these challenges in more detail in Section 2.2.

1.6 Our contribution

In this paper, we present an FPT-algorithm for TREE CONTAINMENT parameterized by the treewidth of the input network. Our algorithm is one of the first (constructive) FPT-algorithms for a problem in phylogenetics parameterized by treewidth. We believe that this is an important development as the treewidth can be much smaller than other parameters such as reticulation number and level which are easier to work with. We see this algorithm as an important step towards the wide application of treewidth-based methods in phylogenetics.

¹ Commonly, MSOL formulations are used to classify problems as FPT but are considered impractical since the resulting running times are dominated by a tower of exponentials of size bounded in the treewidth.

2 Preliminaries

2.1 Reformulating the problem

A key concept throughout this paper will be *display graphs* [9], which are the graphs formed from the union of a phylogenetic tree and a phylogenetic network by identifying leaves with the same labels. Throughout this paper we will let N_{IN} and T_{IN} denote the respective input network and tree in our instance of TREE CONTAINMENT. The main object of study will be the “display graph” of N_{IN} and T_{IN} . For the purposes of our dynamic programming algorithm, we will often consider graphs that are not exactly this display graph, but may be thought of as roughly corresponding to subgraphs of it (though they are not exactly subgraphs; see [40, Appendix A.1]). In order to incorporate such graphs as well, we will define display graphs in a slightly more general way than that usually found in the literature. In particular, we allow for the two “sides” of a display graph to be disconnected, and for some leaves to belong to one side but not the other.

► **Definition 1** (display graph). *A display graph is a directed acyclic graph $D = (V, A)$, with specified subsets $V_T, V_N \subseteq V$ such that $V_T \cup V_N = V$, satisfying the following properties:*

- *The graph $T := D[V_T]$ is an out-forest;*
- *Every vertex has in- and out-degree at most 2 and total degree at most 3;*
- *Any vertex in $V_N \cap V_T$ has out-degree 0 and in-degree at most 1 in both T and $N := D[V_N]$. Herein, we call T the tree side and N the network side of D and we will use the term $D(N, T)$ to denote a display graph with network side N and tree side T .*

Given a phylogenetic network N_{IN} and phylogenetic tree T_{IN} with the same leaf-label set, we define $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ to be the display graph formed by taking the disjoint union of N_{IN} and T_{IN} and identifying pairs of leaves that have the same label. We note that, while the leaves of N_{IN} and T_{IN} were originally labelled, this labelling does not appear in $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$. Labels were used to construct $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$, but in the rest of the paper we will not need to consider them. Indeed, such labels are relevant to the TREE CONTAINMENT problem only insofar as they establish a relation between the leaves of T_{IN} and N_{IN} , and this relation is now captured by the structure of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$.

We now reformulate the TREE CONTAINMENT problem in terms of an *embedding function* on a display graph. Unlike the standard definition of an embedding function (see, e.g., [29]), which is defined for a phylogenetic network N and tree T , our definition of an embedding function applies directly to the display graph $D(N, T)$. Because of our more general definition of display graphs, our definition of an embedding function will also be more general than that found in the literature. The key idea of an embedding function remains the same, however: it shows how a subdivision of T may be viewed as a subgraph of N .

► **Definition 2** (embedding function). *Let D be a display graph with network side N and tree side T , and let $\mathcal{P}(N)$ denote the set of all directed paths in N . An embedding function on D is a function $\phi : V(T) \cup A(T) \rightarrow V(N) \cup \mathcal{P}(N)$ such that:*

- (a) *for each $u \in V(T)$, $\phi(u) \in V(N)$ and, for each arc $uv \in A(T)$, $\phi(uv)$ is a directed $\phi(u)$ - $\phi(v)$ -path in N ;*
- (b) *for any distinct $u, v \in V(T)$, $\phi(u) \neq \phi(v)$;*
- (c) *for any $u \in V(T) \cap V(N)$, $\phi(u) = u$;*
- (d) *the paths $\{\phi(uv) \mid uv \in A(T)\}$ are arc-disjoint;*
- (e) *for any distinct $p, q \in A(T)$, $\phi(p)$ and $\phi(q)$ share a vertex z only if p and q share a vertex w with $z = \phi(w)$;*

Note that the standard definition of an embedding of a phylogenetic tree T into a phylogenetic network N (see e.g. [29]) coincides with the definition of an embedding function on $D(N, T)$. Property (e) ensures that, while the embeddings of arcs uv, vw_1, vw_2 can all meet at $\phi(v)$, the embeddings of different tree arcs cannot otherwise meet. (In particular, the path $\phi(uv)$ cannot end at a reticulation that is also an internal vertex of $\phi(u'v')$, something that is otherwise allowed by properties (a)–(d).)

► **Lemma 3.** *A phylogenetic network N displays a phylogenetic tree T if and only if there is an embedding function on $D(N, T)$.*

In light of Lemma 3, we may henceforth view TREE CONTAINMENT as the following problem:

TREE CONTAINMENT (TC)

Input: phylogenetic network N_{IN} and phylogenetic tree T_{IN} with the same leaf-label set
Task: Find an embedding function on $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$.

2.2 Overview of our approach

We study TREE CONTAINMENT parameterized by the treewidth of the input network N_{IN} . A key tool will be the following theorem.

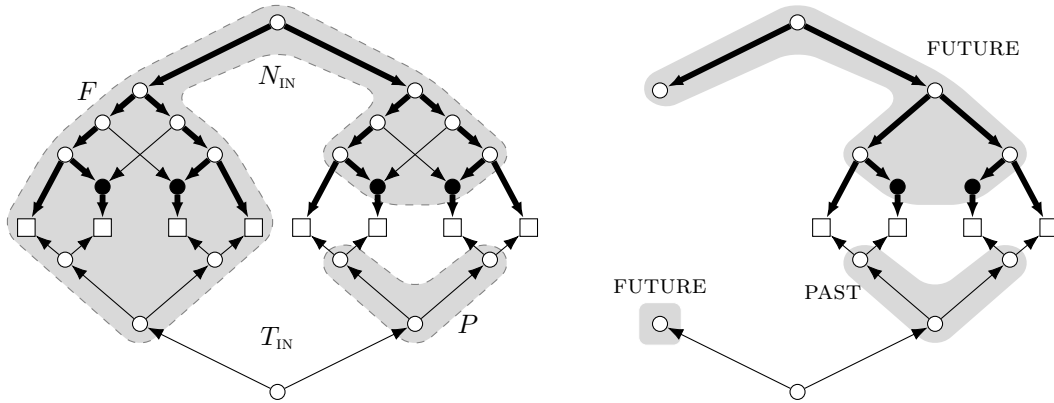
► **Theorem 4** ([30]). *Let N and T be an unrooted binary phylogenetic network and tree, respectively, with the same leaf-label set. If N displays T then $tw(D(N, T)) \leq 2tw(N) + 1$.*

By Theorem 4, we suppose that the display graph $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ has treewidth at most $2k + 1$, where k is the treewidth of the underlying undirected graph N of N_{IN} as, otherwise, N does not display the unrooted version T of T_{IN} , implying that N_{IN} does not display T_{IN} .

As is often the case for treewidth parameterizations, we will proceed via a dynamic programming on a tree decomposition, in this case a tree decomposition of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$. Recall that we view a bag (P, S, F) in the tree decomposition as partitioning the vertices of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ into past, present and future. A typical dynamic programming approach is to store, for each bag, some set of information about the present, while forgetting most information about the past, and not yet caring about what happens in the future. The resulting information is stored in a “signature”, and the algorithm works by calculating which signatures are possible on each bag, in a bottom-up manner. This approach is complicated by the fact that the sought-for embedding of T_{IN} into N_{IN} may not map the past/present/future of T_{IN} into the past/present/future (respectively) of N_{IN} . Vertices from the past of T_{IN} may be embedded in the future of N_{IN} , or vice versa. Thus, we have to store more information than we might at first think. In particular, it is not enough to store information about which present vertices of T_{IN} are embedded in which present vertices of N_{IN} (indeed, depending on the bag, it may be that none of them are). As such, our notion of a “signature” has to track how vertices from the past of T_{IN} are embedded in the present and future of N_{IN} , and which vertices from the past of N_{IN} contain vertices from the present or future of T_{IN} . Vertices of the past which are mapped to vertices of the past, on the other hand, can mostly be forgotten about.

2.3 An informal guide to (compact) signatures

Roughly speaking, a *signature* σ for a bag (P, S, F) in the tree decomposition of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ consists of the following items (see Figure 3 for an example):



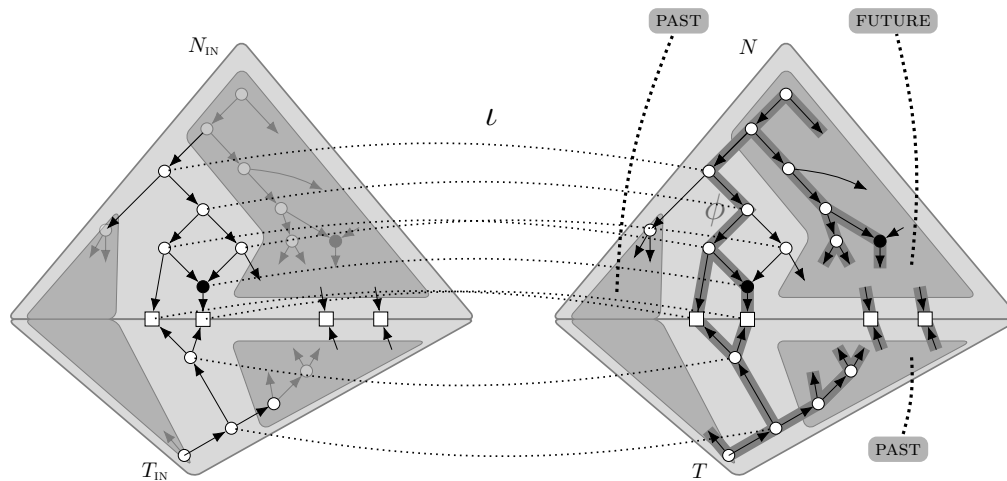
■ **Figure 2 Left:** An example of a display graph $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ for which N_{IN} displays T_{IN} as witnessed by the embedding function ϕ that is indicated by bold edges. Highlighting with dashed border represents the sets P and F , for some bag (P, S, F) in a tree decomposition of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$. **Right:** A representation of the (compact) signature for (P, S, F) derived from this solution. Vertices labelled PAST or FUTURE are highlighted in gray without border.

1. a display graph $D(N, T)$, some of whose vertices correspond (isomorphically) to $S \subseteq V(D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}}))$, and the rest of which are labeled PAST or FUTURE (which we may think of as vertices corresponding to some vertex of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ in P or F , respectively). We use a function ι on $V(D(N, T))$ to capture both this correspondence and labelling, where ι maps each vertex to an element of S or a label from $\{\text{PAST}, \text{FUTURE}\}$.
2. an embedding ϕ of T in N such that, for no arc uv , all of $V(\phi(uv)) \cup \{u, v\}$ have the same label $y \in \{\text{PAST}, \text{FUTURE}\}$ under ι .

Signatures may be seen as “partial embedding functions” on parts of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ in a straightforward way. In particular, we call σ *valid* for (P, S, F) if, roughly speaking, ϕ corresponds (via ι) to something that can be extended to an embedding function on the subgraph of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ induced by the vertices $P \cup S$ introduced below (P, S, F) . In our dynamic programming algorithm, we build valid signatures for a bag x from valid signatures of the child bag(s) of x (in particular, validity for x is implied by validity for the child bag(s)).

Since iterating over all signatures for a bag (P, S, F) (in order to check their validity) exceeds FPT time, we will instead consider “compact” signatures, whose number and size are bounded in the width $|S|$ of the bag (P, S, F) . If $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ admits an embedding function ϕ^* , then a compact signature corresponding to this embedding function exists. In the following, we informally describe the compaction process for this hypothetical solution ϕ^* , thus giving a rough idea of the definition of a “compact” signature. At all times, the (tentative) signature will contain a display graph $D(N, T)$ (initially $D(N, T) = D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$), and an embedding function of T into N (initially ϕ^*). For a more complete description of our approach, see Appendix A in [40], for the proofs and remaining details, see Appendix B, and for an illustration, see Figure 2.

Step 1 After initialization with ϕ^* , we assign a label FUTURE to all vertices of F , and a label PAST to all vertices in P (Observe that no vertex labeled PAST will be adjacent to a vertex labeled FUTURE, since S separates P from F in $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$). Then, we “forget” which vertices of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ the vertices labelled PAST or FUTURE correspond to. Our preliminary signature now contains (1) a display graph $D(N, T)$ whose vertices are either labelled FUTURE or PAST or correspond (isomorphically) to vertices in $S \subseteq V(D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}}))$ (we refer the reader to [40, Appendix A.1] for a more formal description), as well as (2) an embedding function for $D(N, T)$ into N .



■ **Figure 3** Example of a signature of a bag (P, S, F) . The S -part of $D(N_{\text{IN}}, T_{\text{IN}})$ is solid while the non- S part is faded. The embedding ϕ (right, indicated with gray edge-highlight) maps T into N . The dotted arcs labelled ι show the isomorphism between part of $D(N, T)$ and $S \subseteq V(D(N_{\text{IN}}, T_{\text{IN}}))$. Note that the part of $D(N, T)$ that is not mapped to S is not necessarily isomorphic to anything in $D(N_{\text{IN}}, T_{\text{IN}})$.

Step 2 We now simplify the structure of the preliminary signature. The main idea is that, if a is an arc of T with both endpoints labelled PAST and all vertices in the path $\phi(a)$ are also labelled PAST, then we can safely forget a and all the arcs in $\phi(a)$. Intuitively, the information that a will be embedded in $\phi(a)$ does not have any effect on the possible solutions one could construct on the part of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ that is “above” the bag (P, S, F) . Similarly, we can forget any arc a of T whose endpoints, as well as every vertex in $\phi(a)$, are assigned the label FUTURE. Intuitively, this is because this information should have no bearing on whether a solution exists with this signature for $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ restricted to $P \cup S$. In a similar way, we forget any vertex $u \in V(T)$ and its embedding $\phi(u)$ if they are assigned the same label, provided that all their incident arcs can also be forgotten. We will call the vertices and arcs fulfilling these conditions “redundant” and we remove them from our tentative signature. We can also safely delete the vertices and arcs of N that are labelled $y \in \{\text{PAST}, \text{FUTURE}\}$ but are not part of the image of ϕ . As a result, we now have that for any remaining vertex $u \in T$, either one of $\{u, \phi(u)\}$ is labelled PAST and the other labelled FUTURE, or some vertex element of S must appear either one of $\{u, \phi(u)\}$, a neighbor of u , or a vertex in the path $\phi(a)$ for an incident arc a of u . Thus, we have “forgotten” all the aspects of the embedding except those that involve vertices from the present in some way, or those where the embedding “time-travels” between the past and future (see [40, Appendix A.3] for a more formal description of this process).

Step 3 Finally, we may end up with long paths of vertices with in-degree and out-degree 1 that are labelled PAST or FUTURE in N (for example, if u and v are labelled PAST, then $\phi(uv)$ may be a long path in N with all vertices labelled FUTURE). Such long paths do not contain any useful information to us, we therefore compress these by suppressing vertices with in-degree and out-degree 1 (This gives the *compact signature*, see [40, Appendix A.8]).

2.4 Bounding the number of signatures

We now outline the main arguments for why the number of possible (compact) signatures for a given bag (P, S, F) can be bounded in $|S|$. Such a bound on the number of signatures ensures that the running time of the algorithm is FPT, because the number of calculations required for each bag is bounded by a function of the treewidth.

The main challenge is to bound the size of the display graph $D(N, T)$ in a given signature for (P, S, F) . Once such a bound is achieved, this immediately implies upper bounds (albeit quite large) for the number of possible display graphs and the number of possible embeddings, and hence on the number of possible signatures. We will focus here on bounding the size of the tree part T . Once a bound is found for $|T|$ it is relatively straightforward to use that to give a bound on $|N|$ (because the arcs of N that are not used by the embedding of T into N are automatically deleted, unless they are themselves incident to a vertex in S , and because isolated vertices are deleted and long paths suppressed).

It can be seen that a vertex $u \in V(T)$ is redundant (and so would be deleted from the signature) unless one of the following properties holds:

- (1) $u \in S$,
- (2) $\phi(u) \in S$,
- (3) u is incident to an element of S
- (4) for some arc a incident to u , the path $\phi(a)$ contains a vertex from S or
- (5) u and $\phi(u)$ have different labels from $\{\text{PAST}, \text{FUTURE}\}$.

Essentially if none of (1)–(4) holds, then all the vertices mentioned in those properties have the same label as either u or $\phi(u)$, using the fact that S separates the vertices labelled PAST from the vertices labelled FUTURE. If u and $\phi(u)$ have the same label, then all these vertices have the same label, which is enough to show that u is redundant. It remains to bound the number of vertices satisfying one of these properties. For the first four properties, it is straightforward to find a bound in terms of $|S|$. The vertices satisfying the final property are “time-travelling” (in the sense that either u is labelled PAST and $\phi(u)$ FUTURE, or u is labelled FUTURE and $\phi(u)$ PAST). Because of the bounds on the other types of vertices, it is sufficient to provide a bound on the number of *lowest* time-travelling vertices in T .

To see the intuition why this bound should hold: consider some full solution on the original input, i.e. an embedding function on $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$, and suppose $u \in V(T_{\text{IN}})$ is a lowest tree vertex for which $u \in P, \phi(u) \in F$ (thus in the corresponding signature, u has label PAST and $\phi(u)$ has label FUTURE). Let $x \in V(N_{\text{IN}}) \cap V(T_{\text{IN}})$ be some leaf descendant of u . Then there is path in T_{IN} from u to x , and a path in N_{IN} from $\phi(u)$ to $\phi(x) = x$. Thus $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ has an (undirected) path from u to $\phi(u)$. As this is a path between a vertex in P and a vertex in F , some vertex on this path must be in S (since S separates P from F). Such a path must exist for every lowest time-travelling vertex u , and these paths are distinct. The existence of these paths can then be used to bound the number of lowest time-travelling vertices.

3 Algorithm and running time

The final algorithm first computes (or constant-factor approximates) the treewidth of the display graph $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ and concludes non-containment if this computation already implies $tw(D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})) > 2tw(N_{\text{IN}}) + 1$. Otherwise, we proceed with a bottom-up dynamic programming on a nice low-width tree-decomposition, which computes for each bag $x = (P, S, F)$ a set CV_x of “compact-valid signatures” for x (see Section 2.3 for a rough definition). We use “compact-restrictions” to convert a compact signature of one bag into a compact

signature for a different bag. Basically, such a restriction works by mapping certain vertices to a label PAST or FUTURE, removing redundant parts of the display graph and collapsing long paths (see Section 2.3; also see [40, Appendix A.3] for the formal definition).

Leaf bag. If x is a leaf bag, then $P = S = \emptyset$ and all compact signatures $\sigma = (D(N, T), \phi, \iota)$ for x with $\iota^{-1}(\text{PAST}) = \emptyset$ are valid for x (and, thus, included in CV_x).

Introduce- z bag. If x is an introduce bag with child $y = (P, S \setminus \{z\}, F \cup \{z\})$ in \mathcal{T} , then all compact signatures σ for x whose compact- $\{z \rightarrow \text{FUTURE}\}$ -restriction is valid for y (that is, contained in CV_y) are valid for x .

Forget- z bag. If x is a forget bag with child $y = (P \setminus \{z\}, S \cup \{z\}, F)$ in \mathcal{T} , then all compact- $\{z \rightarrow \text{PAST}\}$ -restrictions of compact-valid signatures σ for y are valid for x .

Join bag. For join bags, we use “reconciliations” which are, basically, 3-way analogues of signatures, using labels {LEFT, RIGHT, FUTURE} instead of {PAST, FUTURE} (see [40, Appendix A.7]). In particular, if x is a join bag with children $y_L = (L, S, R \cup F)$ and $y_R = (R, S, L \cup F)$, then we compute all compact reconciliations μ for x and check whether

- the compact- $\{\text{LEFT} \rightarrow \text{PAST}, \text{RIGHT} \rightarrow \text{FUTURE}\}$ -restriction of μ is valid for y_L and
- the compact- $\{\text{RIGHT} \rightarrow \text{PAST}, \text{LEFT} \rightarrow \text{FUTURE}\}$ -restriction of μ is valid for y_R .

Then, the compact- $\{\{\text{LEFT}, \text{RIGHT}\} \rightarrow \text{PAST}\}$ -restriction of each μ verifying these conditions is valid for x .

Correctness

The correctness of the computation of the sets CV_x follows from [40, Lemmas 15–17 and 19]. After having computed CV_x for the root bag r of the decomposition, we conclude that N_{IN} displays T_{IN} if and only if there is a compact-valid signature $(D(N, T), \phi, \iota)$ for the root bag with $\iota^{-1}(\text{FUTURE}) = \emptyset$. The correctness of this follows from [40, Lemma 10].

Running Time

To show that the running time is bounded in a function in the treewidth of N , the main challenge is to bound the number of compact signatures for a bag (P, S, F) by a function of $|S|$ (which, by Theorem 4, we may assume is at most $2tw(N) + 1$). In order to do this, we first bound the size of the display graph $D(N, T)$ in a signature by a function of $|S|$, straightforwardly implying bounds on the number of possible display graphs, embedding functions and isolabellings. Full details are given in [40]; to give a flavor of the proofs, we present the argument bounding the number of arcs in T .

► **Lemma 5.** *Any compact signature $(D(N, T), \phi, \iota)$ for a bag (P, S, F) has $|A(T)| \leq 6|S|$.*

Proof. Let A_S contain all arcs uv of $D(N, T)$ with $\iota(u) \in S$ or $\iota(v) \in S$. As there is only one vertex u with $\iota(u) = s$ for each $s \in S$ and every vertex in $D(N, T)$ has total degree at most 3, we have that $|A_S| \leq 3|S|$. As $\phi(uv)$ and $\phi(u'v')$ are arc-disjoint for any distinct tree arcs uv and $u'v'$, there are at most $|A_S \cap A(N)|$ arcs uv of T for which $\phi(uv)$ contains an arc in A_S . Further, at most $|A_S \cap A(T)|$ arcs in T are incident with a vertex in $\iota^{-1}(S)$. Thus, there are at most $|A_S|$ arcs uv in T for which $\{u, v\} \cup V(\phi(uv))$ contains a vertex of $\iota^{-1}(S)$.

Every remaining arc uv in T has $\iota(\{u, v\} \cup V(\phi(uv))) \subseteq \{\text{PAST}, \text{FUTURE}\}$. Further, we may assume the signature to be “well-behaved” (see [40, Sections A.4 and B.4]), which implies among other things that vertices mapped to PAST and FUTURE cannot be adjacent in $D(N, T)$, and that no arcs or vertices are redundant. Then we have $\iota(u) = \iota(v)$ and $\iota(u') = \iota(v')$ for every arc $u'v'$ in the path $\phi(uv)$. Then, for all but at most $|A_S| \leq 3|S|$ arcs uv of T , we have $\iota(u) = \iota(v) \in \{\text{PAST}, \text{FUTURE}\}$ as well as one of $\iota(V(\phi(uv))) = \{\text{PAST}\}$

and $\iota(V(\phi(uv))) = \{\text{FUTURE}\}$. If $\iota(u) = \iota(v) = \text{PAST}$ and $\iota(V(\phi(uv))) = \{\text{PAST}\}$, then uv is redundant w.r.t. $\{\text{PAST}\}$, a contradiction, and, similarly in case $\iota(u) = \iota(v) = \text{FUTURE}$ and $\iota(\phi(uv)) = \{\text{FUTURE}\}$. So, for all but at most $3|S|$ tree arcs uv , either $\iota(u) = \iota(v) = \text{PAST}$ and $\iota(\phi(uv)) = \{\text{FUTURE}\}$ or $\iota(u) = \iota(v) = \text{FUTURE}$ and $\iota(\phi(uv)) = \{\text{PAST}\}$. In particular, we have that $\iota(\phi(v)) \neq \iota(v)$, and for such vertices we may assume v has out-degree 2 (see Definition in [40, Appendix A.2]). Hence, any lowest arc uv in T is one of the at most $|A_S|$ many arcs uv for which $\{u, v\} \cup V(\phi(uv))$ contain a vertex of $\iota^{-1}(S)$. Thus, in total, T has at most $2|A_S| \leq 6|S|$ arcs. \blacktriangleleft

► **Theorem 6.** *TREE CONTAINMENT can be solved in $2^{O(tw(N_{IN})^2)} \cdot |A(N_{IN})|$ time.*

4 Future work

Before implementing our dynamic programming algorithm, one should first try to reduce the constant in the bound on the number of possible signatures as much as possible. Such reductions may be possible for instance by imposing further structural constraints on the signatures that need to be considered. If this constant can be reduced, possibly including heuristic improvements, it would be interesting to implement the algorithm and test it on practical data.

From a theoretical point of view, there are many opportunities for future work. First, there are multiple variants and generalizations of TREE CONTAINMENT that deserve attention: non-binary inputs, unrooted inputs and inputs consisting of two networks. Indeed, in order to decide if a network is contained in a second network, our approach would have to be extensively modified, since our size-bound on the signatures heavily relies on T_{IN} being a tree.

Second, a major open problem is whether the HYBRIDIZATION NUMBER problem is FPT with respect to the treewidth of the output network. Again there are different variants: rooted and unrooted, binary and non-binary, a fixed or unbounded number of input trees. For some applications, the definition of an embedding has to be relaxed (allowing, for example, multiple tree arcs embedded into the same network arc) [26, 25]. Other interesting candidate problems for treewidth-based algorithms include phylogenetic network drawing [35], orienting phylogenetic networks [27] and phylogenetic tree inference with duplications [39].

Finally, we believe that the approach taken in this paper (applying dynamic programming techniques on a tree decomposition of single graph representing all the input data, with careful attention given to the interaction between past and future) could potentially have applications outside of phylogenetics, in any context where the input to a problem consists of two or more distinct partially-labelled graphs that need to be reconciled.

References

- 1 Julien Baste, Christophe Paul, Ignasi Sau, and Scornavacca Celine. Efficient FPT algorithms for (strict) compatibility of unrooted phylogenetic trees. *Bulletin of Mathematical Biology*, 79:920–938, 2017.
- 2 Matthias Bentert, Josef Malík, and Mathias Weller. Tree containment with soft polytomies. In *SWAT'18*, volume 101, pages 9–1, 2018.
- 3 Vincent Berry, Celine Scornavacca, and Mathias Weller. Scanning phylogenetic networks is NP-hard. In *SOFSEM'20*, pages 519–530. Springer, 2020.
- 4 Hans L Bodlaender. Dynamic programming on graphs with bounded treewidth. In *ICALP'88*, pages 105–118. Springer, 1988.
- 5 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

- 6 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015.
- 7 Hans L Bodlaender, Pål Grønås Drange, Markus S Dregi, Fedor V Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.
- 8 Magnus Bordewich and Charles Semple. Computing the minimum number of hybridization events for a consistent evolutionary history. *Discrete Applied Mathematics*, 155(8):914–928, 2007.
- 9 David Bryant and Jens Lagergren. Compatibility of unrooted phylogenetic trees is FPT. *Theoretical Computer Science*, 351(3):296–302, 2006. Parameterized and Exact Computation.
- 10 Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations*, pages 313–400. World Scientific, 1997.
- 11 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.
- 12 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Joham MM van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS'11*, pages 150–159. IEEE, 2011.
- 13 Eduard Eiben, Robert Ganian, Thekla Hamm, and O-joung Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. *Journal of Computer and System Sciences*, 121:57–75, 2021.
- 14 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.
- 15 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. *Hitting Topological Minors is FPT*, pages 1317–1326. Association for Computing Machinery, New York, NY, USA, 2020.
- 16 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.
- 17 Philippe Gambette, Andreas D. M. Gunawan, Anthony Labarre, Stéphane Vialette, and Louxin Zhang. Solving the tree containment problem for genetically stable networks in quadratic time. In *IWOCA'15*, pages 197–208, 2015.
- 18 Philippe Gambette, Andreas DM Gunawan, Anthony Labarre, Stéphane Vialette, and Louxin Zhang. Solving the tree containment problem in linear time for nearly stable phylogenetic networks. *Discrete Applied Mathematics*, 246:62–79, 2018.
- 19 Robert Ganian, Petr Hliněný, Joachim Kneis, Daniel Meister, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? *Journal of Combinatorial Theory, Series B*, 116:250–286, 2016.
- 20 A Grigoriev, S Kelk, and L Lekic. On low treewidth graphs and supertrees. *Journal of Graph Algorithms and Applications*, 19(1):325–343, 2015.
- 21 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *STOC'11*, pages 479–488, 2011.
- 22 Andreas DM Gunawan. Solving the tree containment problem for reticulation-visible networks in linear time. In *AlCoB'18*, pages 24–36. Springer, 2018.
- 23 Andreas DM Gunawan, Bingxin Lu, and Louxin Zhang. A program for verification of phylogenetic network models. *Bioinformatics*, 32(17):i503–i510, 2016.
- 24 Andreas DM Gunawan, Hongwei Yan, and Louxin Zhang. Compression of phylogenetic networks and algorithm for the tree containment problem. *Journal of Computational Biology*, 26(3):285–294, 2019.
- 25 Katharina T Huber, Simone Linz, and Vincent Moulton. The rigid hybrid number for two phylogenetic trees. *Journal of Mathematical Biology*, 82(5):1–29, 2021.

- 26 Katharina T Huber, Vincent Moulton, Mike Steel, and Taoyang Wu. Folding and unfolding phylogenetic trees and networks. *Journal of Mathematical Biology*, 73(6):1761–1780, 2016.
- 27 Katharina T Huber, Leo van Iersel, Remie Janssen, Mark Jones, Vincent Moulton, Yukihiro Murakami, and Charles Semple. Rooting for phylogenetic networks. *arXiv preprint*, 2019. [arXiv:1906.07430](#).
- 28 Leo van Iersel, Steven Kelk, Georgios Stamoulis, Leen Stougie, and Olivier Boes. On unrooted and root-uncertain variants of several well-known phylogenetic network problems. *Algorithmica*, 80(11):2993–3022, 2018.
- 29 Leo van Iersel, Charles Semple, and Mike Steel. Locating a tree in a phylogenetic network. *Information Processing Letters*, 110(23):1037–1043, 2010.
- 30 R. Janssen, M. Jones, S. Kelk, G. Stamoulis, and T. Wu. Treewidth of display graphs: bounds, brambles and applications. *Journal of Graph Algorithms and Applications*, 23:715–743, 2019.
- 31 Remie Janssen and Yukihiro Murakami. On cherry-picking and network containment. *Theoretical Computer Science*, 856:121–150, 2021.
- 32 Iyad A Kanj, Luay Nakhleh, Cuong Than, and Ge Xia. Seeing the trees and their branches in the network is hard. *Theoretical Computer Science*, 401(1-3):153–164, 2008.
- 33 Steven Kelk, Georgios Stamoulis, and Taoyang Wu. Treewidth distance on phylogenetic trees. *Theoretical Computer Science*, 731:99–117, 2018.
- 34 Steven Kelk, Leo van Iersel, Celine Scornavacca, and Mathias Weller. Phylogenetic incongruence through the lens of monadic second order logic. *Journal of Graph Algorithms and Applications*, 20(2):189–215, 2016.
- 35 Jonathon Klavitter and Peter Stumpf. Drawing tree-based phylogenetic networks with minimum number of crossings. *arXiv preprint*, 2020. [arXiv:2008.08960](#).
- 36 Ton Kloks. *Treewidth: computations and approximations*. Springer, 1994.
- 37 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *FOCS'21*, pages 184–192, 2021.
- 38 Celine Scornavacca and Mathias Weller. Treewidth-based algorithms for the small parsimony problem on networks. In *WABI'21*, 2021.
- 39 Leo van Iersel, Remie Janssen, Mark Jones, Yukihiro Murakami, and Norbert Zeh. Polynomial-time algorithms for phylogenetic inference problems involving duplication and reticulation. *IEEE/ACM transactions on computational biology and bioinformatics*, 17(1):14–26, 2019.
- 40 Leo van Iersel, Mark Jones, and Mathias Weller. Embedding phylogenetic trees in networks of low treewidth. *arXiv preprint*, 2022. [arXiv:2207.00574v2](#).
- 41 Leo van Iersel, Steven Kelk, Nela Lekic, Chris Whidden, and Norbert Zeh. Hybridization number on three rooted binary trees is ept. *SIAM Journal on Discrete Mathematics*, 30(3):1607–1631, 2016.
- 42 Leo van Iersel, Steven Kelk, and Celine Scornavacca. Kernelizations for the hybridization number problem on multiple nonbinary trees. *Journal of Computer and System Sciences*, 82(6):1075–1089, 2016.
- 43 Leo van Iersel and Simone Linz. A quadratic kernel for computing the hybridization number of multiple trees. *Information Processing Letters*, 113(9):318–323, 2013.
- 44 Mathias Weller. Linear-time tree containment in phylogenetic networks. In *RECOMB CG'18*, pages 309–323. Springer, 2018.

Vertex Sparsifiers for Hyperedge Connectivity

Han Jiang ✉

University of Michigan, Ann Arbor, MI, USA

Shang-En Huang ✉

University of Michigan, Ann Arbor, MI, USA

Thatchaphol Saranurak ✉

University of Michigan, Ann Arbor, MI, USA

Tian Zhang ✉

University of Michigan, Ann Arbor, MI, USA

Abstract

Recently, Chalermsook et al. [SODA'21] introduces a notion of *vertex sparsifiers for c -edge connectivity*, which has found applications in parameterized algorithms for network design and also led to exciting dynamic algorithms for c -edge st-connectivity [Jin and Sun FOCS'22].

We study a natural extension called *vertex sparsifiers for c -hyperedge connectivity* and construct a sparsifier whose size matches the state-of-the-art for normal graphs. More specifically, we show that, given a hypergraph $G = (V, E)$ with n vertices and m hyperedges with k terminal vertices and a parameter c , there exists a hypergraph H containing only $O(kc^3)$ hyperedges that preserves all minimum cuts (up to value c) between all subset of terminals. This matches the best bound of $O(kc^3)$ edges for normal graphs by [Liu'20]. Moreover, H can be constructed in almost-linear $O(p^{1+o(1)} + n(rc \log n)^{O(rc)} \log m)$ time where $r = \max_{e \in E} |e|$ is the rank of G and $p = \sum_{e \in E} |e|$ is the total size of G , or in $\text{poly}(m, n)$ time if we slightly relax the size to $O(kc^3 \log^{1.5}(kc))$ hyperedges.

2012 ACM Subject Classification Theory of computation \rightarrow Sparsification and spanners

Keywords and phrases Vertex sparsifier, hypergraph, connectivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.70

Related Version Full Version: <https://arxiv.org/abs/2207.04115>

Acknowledgements We thank Sorrachai Yingchareonthawornchai, Yang Liu, Yunbum Kook, and Richard Peng for the discussion that inspires the notion of the pruned auxiliary graph in this paper. We also thank anonymous reviewers for their valuable comments.

1 Introduction

Graph sparsification has played a central role in graph algorithm research in the last two decades. Prominent examples include spanners [1], cut sparsifiers [3], and spectral sparsifiers [22]. Recently, there has been significant effort in generalizing the graph sparsification results to hypergraphs. For cut sparsifiers, Kogan and Krauthgamer [13] generalized the Benczúr and Karger's cut sparsifiers [3] by showing that, given any hypergraph $G = (V, E)$ with n vertices, there is a $(1+\epsilon)$ -approximate cut sparsifier H containing $\tilde{O}(nr/\epsilon^2)$ hyperedges where $r = \max_{e \in E} |e|$ denotes the *rank* of the hypergraph. After some follow-up work [5, 2], Chen, Khanna, and Nagda [6] finally improved the sparsifier size to $\tilde{O}(n/\epsilon^2)$ hyperedges, matching the optimal bound for normal graphs. Another beautiful line of work generalizes Spielman and Teng's spectral sparsifiers [22] to hypergraphs [2, 21, 10] and very recently results in spectral sparsifiers with $\tilde{O}(n/\text{poly}(\epsilon))$ hyperedges [11]. We also mention that the classical sparse connectivity certificates by Nagamochi and Ibaraki [19] were also generalized to hypergraphs by Chekuri and Xu [5].

This paper studies a graph sparsification problem recently introduced by Chalermsook et al. [4] called *vertex sparsifiers for c -edge connectivity*. It is closely related to the vertex sparsifiers for edge cuts [15, 12] and vertex cuts [14]. In this problem, we are given an



© Han Jiang, Shang-En Huang, Thatchaphol Saranurak, and Tian Zhang;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 70; pp. 70:1–70:13

Leibniz International Proceedings in Informatics



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

unweighted undirected graph $G = (V, E)$ and a set of terminals $\mathcal{T} \subseteq V$. For any disjoint subsets $A, B \subseteq \mathcal{T}$, let $\text{mincut}_G(A, B)$ denote the size of a minimum (edge-)cut that disconnects A and B . Now, a graph $H = (V_H, E_H)$ with $\mathcal{T} \subseteq V_H$ is a (\mathcal{T}, c) -sparsifier of G if for any disjoint subsets $A, B \subseteq \mathcal{T}$, $\min\{c, \text{mincut}_G(A, B)\} = \min\{c, \text{mincut}_H(A, B)\}$. Basically, H preserves all minimum cut structures between the terminals \mathcal{T} up to the value c . This notion of graph sparsifiers has found interesting applications in offline dynamic algorithms and network design problems [4]. Moreover, the very recent breakthrough on dynamic c -edge st-connectivity by Jin and Sun [9] is also crucially based on dynamic algorithms for maintaining (\mathcal{T}, c) -sparsifiers.

In the original paper by [4], they showed that, for any graph $G = (V, E)$ and terminal set \mathcal{T} of size k , there exists a (\mathcal{T}, c) -sparsifier containing $O(kc^4)$ edges (which can be constructed in $O(m(c \log n)^{O(c)})$ time) and also showed fast algorithms for constructing (\mathcal{T}, c) -sparsifiers of size $k \cdot O(c)^{2c}$ in $mc^{O(c)} \log^{O(1)} n$ time. Then, Liu [16] improved the size bound to $O(kc^3)$ together with polynomial-time algorithms (no exponential dependency on c) for constructing (\mathcal{T}, c) -sparsifiers with $O(kc^3 \log^{1.5} n)$ edges.

A natural question is then whether these results can be extended to hypergraphs. The notion of (\mathcal{T}, c) -sparsifiers itself can be naturally extended to hypergraphs by allowing G and H to be hypergraphs and letting $\text{mincut}_G(A, B)$ denote the value of the minimum hyperedge-cut instead. However, it is conceivable that there might not exist a (\mathcal{T}, c) -sparsifier with $\text{poly}(k, c)$. This bound might require bad dependency on the rank r , for example.

In this paper, we show that the state-of-the-art for normal graphs indeed extend to hypergraphs and we can even slightly improve the bounds:

► **Theorem 1.** *Let $G = (V, E)$ be a hypergraph with n vertices, m hyperedges, rank r and total size p . Let $\mathcal{T} \subseteq V$ be the set of k terminals. There are algorithms for computing the following:*

1. *a (\mathcal{T}, c) -sparsifier H of G with $O(kc^3)$ hyperedges in $O(p^{1+o(1)} + n(rc \log n)^{O(rc)} \log m)$ time, which is almost-linear in the input size when both r and $c = O(1)$, and*
2. *a (\mathcal{T}, c) -sparsifier H of G with $O(kc^3 \log^{1.5}(kc))$ hyperedges in $\text{poly}(m, n)$ time.*

The first result matches the best known bound of $O(kc^3)$ edges for normal graphs [16]. When $r = O(1)$, the first time bound slightly improves the $O(m(c \log n)^{O(c)})$ bound of [4] for normal graphs. The second result removes the exponential dependency on r and c after relaxing the size by a $\log^{1.5}(kc)$ factor. The number of hyperedges in our sparsifier is completely independent from n , while the polynomial time algorithm by Liu [16] gives the size of $O(kc^3 \log^{1.5} n)$. So this implies the first polynomial time construction of sparsifiers of size near-linear in k and independent of n , even for normal graphs.

Open Problems. Can we construct vertex sparsifiers for c -hyperedge connectivity of $k \cdot \text{poly}(c)$ size in near-linear time even when the rank is unbounded? This is a prerequisite to near-linear time algorithms for computing vertex sparsifiers for c -vertex connectivity of $k \cdot \text{poly}(c)$ size. Such a result might lead to dynamic c -vertex st-connectivity algorithm similar to the previous development where a near-linear time construction of vertex sparsifiers for c -edge connectivity leads to a dynamic algorithm for c -edge st-connectivity [9]. As dynamic c -vertex st-connectivity is one of the major open problems in dynamic graph algorithms (known solutions only works for very small $c \leq 3$ [7, 8, 20]), we view this work as a stepping stone towards this goal.

1.1 Technical Challenges

There are two main obstacles that prevent us extending the results of [16, 4] directly from normal graphs to hypergraphs. First, if we follow the divide and conquer framework of Chalermsook et al. [4] in a straightforward way, then we would end up with a much larger (\mathcal{T}, c) -sparsifier with $O(|\mathcal{T}|(rc)^3)$ hyperedges. This is because, in [4], all vertices incident to the boundary edges are declared as new terminal vertices in the recursion. However in our case, each hyperedge may contain r vertices and this yields the dependency of r . To handle this issue, we instead introduce only two *anchor vertices* for each boundary hyperedge. Our divide and conquer framework requires slightly more careful analysis, but this naturally gives a (\mathcal{T}, c) -sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges.

The second obstacle is the near-linear time algorithm, Part (1) of Theorem 1. Chalermsook et al. [4] introduced *auxiliary graphs* and apply the ϕ -SPARSIFY procedure on it to identify all *essential* hyperedges, which roughly are hyperedges that will be kept in the sparsifier. However, there is a subtle small gap in [4]: their ϕ -SPARSIFY procedure could erroneously identify non-essential hyperedges as essential hyperedges. This is explained in more detail in the full version of the paper. This bug results in a much larger (\mathcal{T}, c) -sparsifier. In this paper we fix the bug by (1) introducing a notion of *useful* partitions of the terminal set and (2) providing an efficient algorithm that discards all non-useful partitions from the auxiliary graph. Then, we show that, after our modification, this approach indeed gives a small (\mathcal{T}, c) -sparsifier as desired.

1.2 Organization

In Section 2 we review some basic definitions of hypergraphs. In Section 3 we define contraction based (\mathcal{T}, c) -sparsifiers and introduce the divide and conquer framework. In Section 4 we show the existence of a (\mathcal{T}, c) -sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges. In Section 5 we give a near-linear-time algorithm that computes a (\mathcal{T}, c) -sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges, proving Part (1) of Theorem 1. For Part (2) of Theorem 1 we refer readers to the full version of the paper.

2 Preliminary

Let $G = (V, E)$ be a hypergraph. V is the set of vertices and E is a multiset of hyperedges with each hyperedge e being a subset of V . The *rank* $r := \max_{e \in E} |e|$ of a hypergraph is the size of the largest hyperedge, and the *total size* $p := \sum_{e \in E} |e|$ is the sum of all edge sizes.

For any two disjoint sets of vertices $A, B \subseteq V$, let $E_G(A, B)$ denote the set of hyperedges with at least one endpoint in A and at least one endpoint in B . For any set of vertices $X \subseteq V$, we denote the *boundary* of X of the graph G by $\partial_G X := E_G(X, V \setminus X)$. If the context is clear then we will omit the graph G and write ∂X instead.

Restrictions and Induced Sub-Hypergraphs. Let $\mathcal{T} \subseteq V \cup E$ be a mixed multiset of vertices and hyperedges, for any set of vertices $X \subseteq V$, we define the *restriction* of the multiset \mathcal{T} on X to be $\mathcal{T}|_X = (\mathcal{T} \cap X) \cup \{e \cap X \mid e \in (\mathcal{T} \cap E) \text{ and } e \cap X \neq \emptyset\}$. The *induced sub-hypergraph* $G[X]$ is then defined over the vertex set X with the restriction of all hyperedges $E|_X$, that is, $G[X] := (X, E|_X)$.

Incident Edges and Vertices. For any set of vertices $X \subseteq V$, define $E(X)$ to be the set of all hyperedges that incident to at least one vertex in X . For any set of hyperedges $Y \subseteq E$, define $V(Y) = \bigcup_{y \in Y} y$ to be the set of vertices incident to hyperedges in Y . Similarly, for any mixed set of vertices and hyperedges $\mathcal{T} \subseteq V \cup E$ we define $V(\mathcal{T}) = (\mathcal{T} \cap V) \cup V(\mathcal{T} \cap E)$ to be the set of all vertices that are in the set or incident to any hyperedge in the set.

3 Structural Properties on Hypergraphs

In this section, we explore more structural properties on hypergraphs. In particular, we introduce *anchored separated hyperedges*, and describe useful properties in a divide and conquer framework that leads to a construction of (\mathcal{T}, c) -sparsifiers.

3.1 Cuts in Hypergraphs

Let u and v be two elements in V . We say that u and v are *connected* in a hypergraph G , if there is a path connecting u and v . Let $A, B \subseteq V$ be two disjoint sets of vertices. A and B are *disconnected* if for any $a \in A$ and $b \in B$, a and b are not connected.

► **Definition 2** (Cuts and Minimum Cuts). *A cut is a bipartition $(X, V \setminus X)$ of vertices. The value of the cut is $|\partial X| = |E_G(X, V \setminus X)|$. For any disjoint subsets $A, B \subseteq V$, if $A \subseteq X$ and $B \subseteq (V \setminus X)$ then we say that $(X, V \setminus X)$ is an (A, B) -cut. A minimum (A, B) -cut or (A, B) -mincut is any (A, B) -cut with minimum value. Its value is denoted as $\text{mincut}_G(A, B)$. Given a parameter c , a c -thresholded (A, B) -mincut cut value is defined as*

$$\text{mincut}_G^c(A, B) := \min(\text{mincut}_G(A, B), c).$$

3.2 (\mathcal{T}, c) -Equivalency and (\mathcal{T}, c) -Sparsifiers

Our vertex sparsifier algorithms are based on identifying a set of hyperedges and contract them. Given a hypergraph $G = (V, E)$ and a hyperedge $e \in E$, the *contracted hypergraph* G/e is defined by identifying all incident vertices $V(e)$ as one vertex, and then remove e itself from the graph. For any set of terminals $T \subseteq V$, the effect of contracting an hyperedge e is denoted as T/e . Similarly, for any set $\hat{E} \subseteq E$, we denote G/\hat{E} the hypergraph obtained from G by contracting all hyperedges in \hat{E} (notice that all hyperedges in \hat{E} are removed after the contraction.)

► **Definition 3** ((\mathcal{T}, c) -Sparsifiers). *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two hypergraphs. Let $\mathcal{T} \subseteq V_G$ be the set of terminals. We say H is a contraction based (\mathcal{T}, c) -sparsifier of G , if there exists a surjective (onto) projection $\pi : V_G \rightarrow V_H$, such that for any $e \in E_H$ there is an edge $f \in E_G$ such that $\pi(f) = \bigcup_{v \in f} \{\pi(v)\} = e$, and for any two subsets $T_1, T_2 \subseteq \mathcal{T}$,*

$$\text{mincut}_G^c(T_1, T_2) = \text{mincut}_H^c(\pi(T_1), \pi(T_2)).$$

Furthermore, if the terminals are not affected by the projection, i.e., $\pi(\mathcal{T}) = \mathcal{T}$, then we say that G and H are (\mathcal{T}, c) -equivalent.

Remark. A more general (\mathcal{T}, c) -sparsifier would allow an arbitrary mapping π on both vertices and edges. However, we note that all (\mathcal{T}, c) -sparsifiers constructed in this paper are always contraction based. Therefore, for the ease of the presentation we will omit the term “contraction based” when we mention (\mathcal{T}, c) -sparsifiers.

For the ease of the reading, we define the following set operations that allow us to add/remove hyperedges of G into a sparsifier H .

► **Definition 4.** Let $G = (V, E)$ be a hypergraph. For any multiset X of hyperedges over the vertices V , and any contraction based (\mathcal{T}, c) -sparsifier H with the projection π , define

- (Adding contracted hyperedges) $H \cup X := H \cup \pi(X)$, and
- (Removing contracted hyperedges) $H - X := H - \pi(X)$.

3.3 (\mathcal{T}, c) -Sparsifiers from a Divide and Conquer Framework

Another important concept to our contraction based (\mathcal{T}, c) -sparsifier construction is that we apply a divide and conquer framework to G . Let $G = (V, E)$ be a hypergraph and let (V_1, V_2) be a bipartition of vertices. We note that our divide and conquer framework is slightly different than just recurse on the induced sub-hypergraphs $G[V_1]$ and $G[V_2]$. In particular, for each *separated hyperedge* e we add two new *anchor vertices* to e , ended up slightly increasing the size of the vertex set in the next-level recursion.

Separated Hyperedges and Anchor Vertices. Let $e \in E(V_1, V_2)$ be a hyperedge across the bipartition. The *separated hyperedges* of e with respect to this bipartition (V_1, V_2) is the set composed of hyperedges of e restricted on both V_1 and V_2 . The *anchored separated hyperedges* are separated hyperedges with additional *anchor vertices*: let $e_1 = e|_{V_1}$ and $e_2 = e|_{V_2}$ be the separated hyperedges of e , then we introduce four new anchored vertices $v_{e,1}, v_{e,2}, v_{e,3}$, and $v_{e,4}$ and define $\hat{e}_1 := e_1 \cup \{v_{e,1}, v_{e,2}\}$ and $\hat{e}_2 := e_2 \cup \{v_{e,3}, v_{e,4}\}$. Let $S = E(V_1, V_2)$ be the set of crossing hyperedges, in this paper the set of anchored separated hyperedges respect to bipartition (V_1, V_2) are denoted by $Sep(S, V_1, V_2) := \{\hat{e}_1, \hat{e}_2 \mid e \in S\}$.

Let $A_1 = \{v_{e,1}, v_{e,2} \mid e \in E(V_1, V_2)\}$ and let $A_2 = \{v_{e,3}, v_{e,4} \mid e \in E(V_1, V_2)\}$ be the set of newly introduced anchor vertices. These anchor vertices will be added to the terminal set in order to correctly preserve the mincut values. That is, the terminal sets defined for the subproblems are $\mathcal{T}_1 := \mathcal{T}|_{V_1} \cup A_1$ and $\mathcal{T}_2 := \mathcal{T}|_{V_2} \cup A_2$. Now, we define the anchored induced sub-hypergraphs, which are useful when applying the divide and conquer framework.

► **Definition 5 (Anchored Induced Sub-Hypergraphs).** Let $G = (V, E)$ be a hypergraph and $V_1 \subseteq V$ be a subset of vertices. Define $V_2 = V \setminus V_1$, $G^{\text{sep}} = G \cup Sep(E(V_1, V_2), V_1, V_2) - E(V_1, V_2)$, and the set of anchored vertices to be $A_1 \cup A_2$. Then, the anchored induced sub-hypergraph for V_1 is defined as $\hat{G}[V_1] := G^{\text{sep}}|_{V_1 \cup A_1}$.

The Divide and Conquer Framework. The most generic divide and conquer method works as the follows. First, a bipartition (V_1, V_2) are determined. Then, the algorithm performs recursion on the anchored induced sub-hypergraphs $\hat{G}[V_1]$ and $\hat{G}[V_2]$ with terminal sets \mathcal{T}_1 and \mathcal{T}_2 respectively. After obtaining the (\mathcal{T}_1, c) -sparsifier and (\mathcal{T}_2, c) -sparsifier from the subproblems, the algorithm combines them by replacing the anchored separated hyperedges with the original hyperedges.

■ **Algorithm 1** A Divide and Conquer Framework.

Input: Hypergraph G , terminal set \mathcal{T} , bipartition (V_1, V_2) of vertices, parameter c .

Output: A (\mathcal{T}, c) -sparsifier H for G .

1 **(Divide)** Construct subproblems $(\hat{G}[V_1], \mathcal{T}_1)$ and $(\hat{G}[V_2], \mathcal{T}_2)$.

2 **(Conquer)** For $i \in \{1, 2\}$, obtain H_i , a (\mathcal{T}_i, c) -sparsifier of $\hat{G}[V_i]$.

3 **(Combine)** Return $H := H_1 \cup H_2 \cup E(V_1, V_2) - Sep(E(V_1, V_2), V_1, V_2)$.

We summarize the divide and conquer framework in Algorithm 1. The following lemma states the correctness of the framework.

► **Lemma 6.** H returned from Algorithm 1 is a (\mathcal{T}, c) -sparsifier.

Proof. Let $\pi_1 : V_1 \cup A_1 \rightarrow V_{H_1}$ and $\pi_2 : V_2 \cup A_2 \rightarrow V_{H_2}$ be the projection maps on H_1 and H_2 respectively. Since $V_1 \cup A_1$ and $V_2 \cup A_2$ are disjoint, it is natural to define $\pi : V \rightarrow V_H$ by simply combining both maps where $\pi(v) = \pi_1(v)$ if $v \in V_1$, and $\pi(v) = \pi_2(v)$ if $v \in V_2$.

Now, it suffices to show that for any two disjoint subsets $A, B \subseteq \mathcal{T}$, we have $\text{mincut}_G^c(A, B) = \text{mincut}_H^c(\pi(A), \pi(B))$.

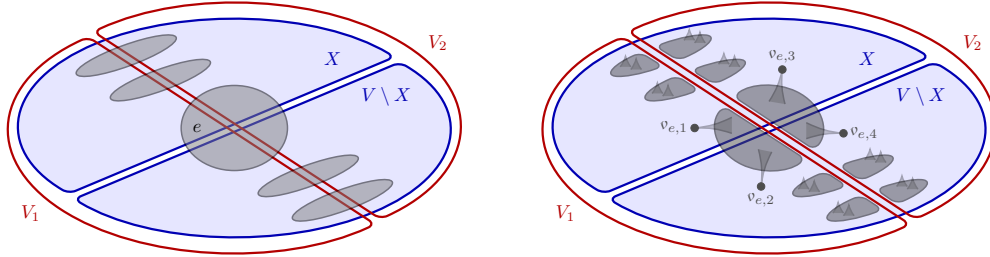
Part 1. We first show that $\text{mincut}_G^c(A, B) \geq \text{mincut}_H^c(\pi(A), \pi(B))$. Let $(X, V \setminus X)$ be a minimum (A, B) -cut on G with size $|\partial X| \leq c$. Intuitively, we will construct the cuts in the subproblems $\hat{G}[V_1]$ and $\hat{G}[V_2]$ using $(X, V \setminus X)$. Then we will argue that the preserved mincuts in $\hat{G}[V_1]$ and $\hat{G}[V_2]$ can be merged back, proving that there is a $(\pi(A), \pi(B))$ -mincut in H with size no larger than $|\partial X|$.

Let $S = E_G(V_1, V_2)$ be the set of hyperedges across the bipartition in the divide and conquer framework, and let $\hat{V} = V \cup A_1 \cup A_2$ be the vertex set in G^{sep} . We define the set of vertices X^{sep} that contains X and all newly created anchor vertices that belongs to the X side: for any $e \in S$, we add $\{v_{e,1}, v_{e,2}, v_{e,3}, v_{e,4}\}$ to X^{sep} if $e \subseteq X$ (the hyperedge is fully in the X side). We add $\{v_{e,1}, v_{e,3}\}$ to X^{sep} if $e \in S$. We add nothing if $e \subseteq V \setminus X$.

Now, we have $\partial X^{\text{sep}} = (\partial X) \cup \text{Sep}((\partial X) \cap S, V_1, V_2) - (\partial X) \cap S$. Moreover, $(X^{\text{sep}}, \hat{V} \setminus X^{\text{sep}})$ is an $(A^{\text{ext}}, B^{\text{ext}})$ -cut in G^{sep} of size $|\partial X| + |(\partial X) \cap S|$, where

$$\begin{cases} A^{\text{ext}} := A \cup \{v_{e,1}, v_{e,3} \mid e \in ((\partial X) \cap S)\}, \text{ and} \\ B^{\text{ext}} := B \cup \{v_{e,2}, v_{e,4} \mid e \in ((\partial X) \cap S)\}. \end{cases}$$

Intuitively, by carefully extend the pair (A, B) to a larger pair $(A^{\text{ext}}, B^{\text{ext}})$ we ensure that *all* separated hyperedges $\text{Sep}((\partial X) \cap S, V_1, V_2)$ appear in every $(A^{\text{ext}}, B^{\text{ext}})$ -mincut on G^{sep} . See Figure 1.



■ **Figure 1** An illustration to the proof of Lemma 6. The gray circles represent hyperedges that cross the bipartition (V_1, V_2) in the divide and conquer framework. When these hyperedges are separated, new anchor vertices are introduced and added to the terminal sets. The newly created terminal vertices are forced to join different sides of the cut, if and only if the separated hyperedge crosses the (A, B) -mincut $(X, V \setminus X)$.

Suppose H_1 is a (\mathcal{T}_1, c) -sparsifier of $\hat{G}[V_1]$ and H_2 is a (\mathcal{T}_2, c) -sparsifier of $\hat{G}[V_2]$ obtained from the conquer step (Algorithm 1). Let $(Y_1, \pi(V_1 \cup A_1) \setminus Y_1)$ and $(Y_2, \pi(V_2 \cup A_2) \setminus Y_2)$ be a $(\pi(A^{\text{ext}}|_{V_1 \cup A_1}), \pi(B^{\text{ext}}|_{V_1 \cup A_1}))$ -mincut on H_1 and a $(\pi(A^{\text{ext}}|_{V_2 \cup A_2}), \pi(B^{\text{ext}}|_{V_2 \cup A_2}))$ -mincut on H_2 respectively. Notice that every hyperedge in $\text{Sep}((\partial X) \cap S, V_1, V_2)$ are in $\partial(Y_1 \cup Y_2)$. Let $Y_0 := Y_1 \cup Y_2$ and after removing all anchor vertices we get $Y = Y_0 \cap V$. Now $(Y, \pi(V) \setminus Y)$ is a $(\pi(A), \pi(B))$ -cut. Since for each hyperedge $e \in (\partial_G X) \cap S$, e is separated into two hyperedges and both of them are in $\partial_{H_1 \cup H_2} Y_0$, we have

$$|\partial_H Y| \leq |\partial_{H_1 \cup H_2} Y_0| - |(\partial_G X) \cap S|. \quad (1)$$

Notice that the inequality in Equation (1) comes from the fact that $\partial_{H_1 \cup H_2} Y_0$ may or may not contain more separated hyperedges from $Sep(S \setminus \partial X, V_1, V_2)$.

Finally we obtain

$$\begin{aligned} \text{mincut}_H^c(\pi(A), \pi(B)) &\leq |\partial_H Y| && (Y \text{ is some } (\pi(A), \pi(B))\text{-cut on } H) \\ &\leq |\partial_{H_1 \cup H_2} Y_0| - |(\partial X) \cap S| && (\text{by Equation (1)}) \\ &= |\partial_{H_1} Y_1| + |\partial_{H_2} Y_2| - |(\partial X) \cap S| \\ &&& (Y_0 \text{ is the disjoint union } Y_1 \cup Y_2) \\ &\leq |\partial_{G^{\text{sep}}} X^{\text{sep}}| - |(\partial X) \cap S| && (X^{\text{sep}} \text{ is some } (A^{\text{ext}}, B^{\text{ext}})\text{-cut}) \\ &= |\partial X| && (\text{exactly } |(\partial X) \cap S| \text{ hyperedges were separated}) \\ &= \text{mincut}_G^c(A, B) && (X \text{ is an } (A, B)\text{-mincut}) \end{aligned}$$

as desired.

Part 2. The proof of $\text{mincut}_H^c(\pi(A), \pi(B)) \geq \text{mincut}_G^c(A, B)$ is very similar to Part 1, so we defer the proof (for completeness) in the full version. ◀

3.4 $(5c, c)$ -Edge-Unbreakable Terminals

Let $G = (V, E)$ be a hypergraph and let $\mathcal{T} \subseteq V$ be the set of terminals. By adopting the notations from [17], we say that a terminal set \mathcal{T} is $(5c, c)$ -edge-unbreakable on G if for any bipartition (V_1, V_2) of V with no more than c crossing edges $|E_G(V_1, V_2)| \leq c$, either $|\mathcal{T}|_{V_1} < 5c$ or $|\mathcal{T}|_{V_2} < 5c$. That is, if there is a cut of size at most c , then at least one of the sides has less than $5c$ induced terminals.

Liu [16] obtained an (\mathcal{T}, c) -sparsifier of size $O(|\mathcal{T}|c^2)$ with a $(5c, c)$ -edge-unbreakable terminal set \mathcal{T} where each terminal vertex $v \in \mathcal{T}$ has degree 1. It turns out that Liu's techniques naturally extend to hypergraphs. We prove the following in the full version of the paper.

► **Lemma 7.** *Let $G = (V, E)$ be a hypergraph and let $\mathcal{T} \subseteq V$ be a set of degree-1 terminals. If \mathcal{T} is $(5c, c)$ -edge-unbreakable on G , then there exists a subset $E' \subseteq E$ with $O(|\mathcal{T}|c^2)$ hyperedges, such that $G/(E - E')$ is a (\mathcal{T}, c) -sparsifier of G .*

4 Existence of (\mathcal{T}, c) -Sparsifiers with $O(kc^3)$ Hyperedges

With all the tools equipped in the previous section, we are able to prove the existence of a (\mathcal{T}, c) -sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges.

► **Theorem 8.** *Let $G = (V, E)$ be a hypergraph and $\mathcal{T} \subseteq V$ be the set of terminals. Then there is a subset $E' \subseteq E$ such that $|E'| = O(|\mathcal{T}|c^3)$ and the contracted hypergraph $G/(E - E')$ is (\mathcal{T}, c) -equivalent to G .*

To prove Theorem 8, it suffices to prove the following Lemma 9 where every terminal vertex has degree 1:

► **Lemma 9.** *Let $G = (V, E)$ be a hypergraph and $\mathcal{T} \subseteq V$ be the set of degree 1 terminals. Then there is a subset $E' \subseteq E$ such that $|E'| = O(|\mathcal{T}|c^2)$ and the contracted hypergraph $G/(E - E')$ is (\mathcal{T}, c) -equivalent to G .*

70:8 Vertex Sparsifiers for Hyperedge Connectivity

Proof of Theorem 8. Without loss of generality, we may assume that each vertex in \mathcal{T} has degree at most c , by duplicating each terminal vertex and add c parallel edges between the duplicated vertex and the original vertex. Let \mathcal{T} be the terminal set of an input instance. Now, assuming each terminal has degree at most c , we can further duplicate each of these terminals c times so we have a set \mathcal{T}' of at most $|\mathcal{T}|c$ degree-1 terminal vertices. By Lemma 9, there exists a subset $E' \subseteq E$ such that $|E'| = O(|\mathcal{T}'|c^2) = O(|\mathcal{T}|c^3)$ and the contracted hypergraph $G/(E - E')$ is (\mathcal{T}, c) -equivalent to G . ◀

To prove Lemma 9, we first present an algorithm SPARSIFYSLOW (See Algorithm 2). The algorithm recursively apply divide and conquer framework until the terminal set is $(5c, c)$ -edge-unbreakable as the base case. After applying Lemma 7 on each base case, the algorithm combines the sparsifiers from the subproblems using Lemma 6.

■ **Algorithm 2** SPARSIFYSLOW SPARSIFYSLOW(G, \mathcal{T}, c).

Input: An undirected unweighted multi-hypergraph G , a set of degree-1 vertex terminals $\mathcal{T} \subseteq V$, and a constant c .

Output: A (\mathcal{T}, c) -sparsifier H for G .

- 1 **if** \mathcal{T} is $(5c, c)$ -edge-unbreakable **then**
- 2 Construct H , a (\mathcal{T}, c) -sparsifier of G using Lemma 7.
- 3 **return** H .
- 4 **else**
- 5 Let (V_1, V_2) be a bipartition of $V(G)$ that refutes the $(5c, c)$ -edge-unbreakable property. That is, $|E_G(V_1, V_2)| \leq c$ but $|\mathcal{T} \cap V_1| \geq 5c$ and $|\mathcal{T} \cap V_2| \geq 5c$.
- 6 Obtain $\begin{cases} H_1 \leftarrow \text{SPARSIFYSLOW}(\hat{G}[V_1], \mathcal{T}_1, c), \text{ and} \\ H_2 \leftarrow \text{SPARSIFYSLOW}(\hat{G}[V_2], \mathcal{T}_2, c). \end{cases}$
- 7 **return** $H \leftarrow H_1 \cup H_2 \cup E(V_1, V_2) - \text{Sep}(S, V_1, V_2)$.
- 8 **end**

Lemma 10 and Lemma 11 give the correctness proof and the size to the returned (\mathcal{T}, c) -sparsifier from Algorithm 2.

► **Lemma 10.** *Algorithm 2 returns a (\mathcal{T}, c) -sparsifier of G .*

Proof. First we notice that all vertices in \mathcal{T}_1 and \mathcal{T}_2 have degree 1 in $\hat{G}[V_1]$ and $\hat{G}[V_2]$ respectively: the anchor vertices have degree 1 and so the recursive calls in Algorithm 2 are valid. The correctness is then recursively guaranteed by Lemma 6 (divide-and-conquer step) and Lemma 7 (base case). ◀

► **Lemma 11.** *Let G be a hypergraph, $\mathcal{T} \subseteq V$ is the set of degree-1 terminal vertices, and let c be a constant. Let $H = \text{SPARSIFYSLOW}(G, \mathcal{T}, c)$ be the output of Algorithm 2. Then H has at most $O(|\mathcal{T}|c^2)$ hyperedges.*

The proof to Lemma 11 is via a potential function similarly defined in Liu [16].

Proof. The execution to Algorithm 2 defines a recursion tree. If $|\mathcal{T}| < 5c$, then the recursion terminates immediately because \mathcal{T} is trivially $(5c, c)$ -edge-unbreakable by definition and a (\mathcal{T}, c) -sparsifier of $O(|\mathcal{T}|c^2)$ hyperedges is returned by Lemma 7. Assume that $|\mathcal{T}| \geq 5c$, then each recursive call on the subproblem (G', \mathcal{T}') guarantees that $|\mathcal{T}'| \geq 5c$.

Now, it suffices to use the following potential function to prove that the total number of terminal vertices in all recursion tree leaves can be bounded by $O(|\mathcal{T}|)$. Define a potential function for each subproblem (G', \mathcal{T}') to be $\Phi(G', \mathcal{T}') := |\mathcal{T}'| - 5c$. Then, according to Algorithm 2, whenever (G', \mathcal{T}') splits into two subproblems $(\hat{G}'[V_1], \mathcal{T}'_1)$ and $(\hat{G}'[V_2], \mathcal{T}'_2)$ we have

$$\begin{aligned} \Phi(\hat{G}'[V_1], \mathcal{T}'_1) + \Phi(\hat{G}'[V_2], \mathcal{T}'_2) &\leq |\mathcal{T}' \cap V_1| + |\mathcal{T}' \cap V_2| + 4|E_{G'}(V_1, V_2)| - 10c \\ &\leq \Phi(G', \mathcal{T}') - c. \end{aligned}$$

Since every subproblem has a non-negative potential, and the sum of potential decreases by c at each divide-and-conquer step, the total number of leaf cases do not exceed $\Phi(G, \mathcal{T})/c \leq |\mathcal{T}|/c$. Hence, the total size from the base case is at most $\sum_{(G', \mathcal{T}'): \text{base case}} |\mathcal{T}'| \leq \Phi(G, \mathcal{T}) + (5c)(\# \text{ of leaf cases}) = O(|\mathcal{T}|)$.

By Lemma 7, the total number of hyperedges returned from Algorithm 2 is at most $O(|\mathcal{T}|c^2)$. The total number of hyperedges added back at Algorithm 2 is at most the number of divide-and-conquer steps times the cut size, which is at most $|\mathcal{T}|$. Therefore, the output (\mathcal{T}, c) -sparsifier H has at most $O(|\mathcal{T}|c^2)$ hyperedges as desired. \blacktriangleleft

Proof of Lemma 9. Lemma 9 follows immediately after the correctness proof (Lemma 10) and upper bounding the number of hyperedges (Lemma 11). \blacktriangleleft

5 An Almost-linear-time Algorithm Constructing a Sparsifier

This section is devoted to proving part (1) in Theorem 1. That is, we give a almost-linear-time (assuming a constant rank) algorithm that constructs a contraction based (\mathcal{T}, c) -sparsifier of $O(|\mathcal{T}|c^3)$ hyperedges which matches with Theorem 8 up to a constant factor. We summarize the result in Theorem 12.

► Theorem 12. *Let $G = (V, E)$ be a hypergraph with n vertices, m hyperedges, and rank $r = \max_{e \in E} |e|$. Let $\mathcal{T} \subseteq V$ be a terminal set $\mathcal{T} \subseteq V$. Then there exists a randomized algorithm which constructs a (\mathcal{T}, c) -sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges in $O(p + n(rc \log n)^{O(rc)} \log m)$ time.*

Overview of the algorithm. Although Algorithm 2 can construct a (\mathcal{T}, c) -sparsifier with $O(|\mathcal{T}|c^3)$ hyperedges, it is slow because we do not have an efficient algorithm searching for a bipartition that violates the $(5c, c)$ -edge-unbreakable property.

To construct our contraction-based (\mathcal{T}, c) -sparsifier, all we need to do is identifying *essential* hyperedges and contract non-essential ones. Essential hyperedges are indispensable to maintaining mincut between terminals. It seems to be challenging to identify essential hyperedges on an arbitrary graph without a $(5c, c)$ -edge-unbreakable guarantee. Fortunately, we notice there is an efficient way to identify essential hyperedges in an expander.

Naturally, we can utilize EXPANDERDECOMPOSE (where the version for hypergraphs is explicitly stated in [18]) which splits a hypergraph into expanders. Expander decomposition not only guarantees expander sub-hypergraphs, but also fits in the divide-and-conquer framework indicated by Lemma 6 with a favorable almost-linear time. Then, we can focus on identifying essential hyperedges in an expander.

To identify essential hyperedges in an expander, we first enumerate all *connected cuts*¹ with value at most c – the sub-hypergraph induced by the smaller side of a connected cut

¹ In Chalermsook et al. [4], the concept of connected cuts is not explicitly defined. We give a formal definition in the full version and hope it clarifies some ambiguity in their paper.

70:10 Vertex Sparsifiers for Hyperedge Connectivity

is connected; see Algorithm 3 and Algorithm 4. Then, we build a *pruned auxiliary graph* based on the cuts we have enumerated. The pruned auxiliary graph leads to an efficient way identifying essential hyperedges. Finally, we contract all detected non-essential hyperedges. We call the above procedure that sparsifies an expander ϕ -SPARSIFY. See Algorithm 5.

■ Algorithm 3 ENUMERATECUTS (G, ϕ, r, c).

Input: A ϕ -expander hypergraph $G = (V, E)$ with rank r , and a threshold parameter c .

Output: All connected cuts with value at most c .

```

1  $\mathcal{C} \leftarrow \emptyset$ . // Stores all found connected cuts.
2 for each  $v_{seed} \in V$  do
3   | /* Invokes a helper function to find all connected cuts involving  $v_{seed}$ . */
3   |  $\mathcal{C} \leftarrow \mathcal{C} \cup \text{ENUMERATECUTSHELP}(0, G, G, \phi, r, c, v_{seed})$ . // See Algorithm 4.
4 end
5 return  $\mathcal{C}$ .
```

■ Algorithm 4 ENUMERATECUTSHELP ($depth, H, G, \phi, r, c, v_{seed}$).

Input: The current recursion depth $depth$. A hypergraph $H = (V, E)$ with rank r . The original hypergraph G . Parameters c and ϕ . A seed vertex $v_{seed} \in V$.

Output: All connected cut with value at most c so that v_{seed} is in the smaller side.

```

1 if  $depth \leq rc$  then
2   | Run DFS from  $v_{seed}$  on  $H$  and stop as soon as visiting  $c\phi^{-1} + 1$  hyperedges.
3   | Let  $\hat{E}$  be the set of visited hyperedges and  $X$  be the set of visited vertices.
4   | if DFS gets stuck before visiting  $c\phi^{-1} + 1$  hyperedges then
5     |   if  $|\partial_G X| \leq c$  then
6       |     return  $\{(X, V \setminus X)\}$ . /* Some connected cut with value at most  $c$ . */
7     |   else
8       |     return  $\emptyset$ .
9     |   end
10  | else
11  |    $\mathcal{S} \leftarrow \emptyset$ .
12  |   for each  $e \in \hat{E}$  and for each  $v \in e, v \neq v_{seed}$  do
13  |     | Let  $e' \leftarrow e \setminus v$ . /* modify the boundary hyperedge into a smaller one. */
14  |     | // A recursive call with  $v$  being removed from  $e$ .
14  |     |  $\mathcal{S} \leftarrow \mathcal{S} \cup \text{ENUMERATECUTSHELP}(depth + 1, H - e + e', G, \phi, r, c, v_{seed})$ 
15  |     | end
16  |     return  $\mathcal{S}$ .
17  | end
18 else
19 | return  $\emptyset$ .
20 end
```

■ **Algorithm 5** ϕ -SPARSIFY ($G, \mathcal{T}, \phi, r, c$).

Input: ϕ -expander hypergraph $G = (V, E)$ with rank r , terminal set \mathcal{T} , threshold parameter c .

Output: A (\mathcal{T}, c) -sparsifier of G .

- 1 Run ENUMERATECUTS and construct the pruned auxiliary graph $G^{\text{aux}} = (V^{\text{aux}}, E^{\text{aux}})$, where $V^{\text{aux}} = P_0 \cup C_0 \cup E_0$.
- 2 Let $E' \leftarrow E \setminus E_0$.
- 3 **for** each $e \in E_0$ (in any order) **do**
- 4 Compute the set of partitions $P'_e := P_0 \cap N(N(e))$ who has at least one mincut that contains the edge e .
- 5 **if** $\forall p \in P'_e, N(p) \not\subseteq N(e)$ **then**
- 6 Remove $N(e)$ and all incident edges from G^{aux} .
- 7 $E' \leftarrow E' \cup \{e\}$. // e is non-essential.
- 8 **end**
- 9 **end**
- 10 **return** G/E' .

■ **Algorithm 6** SPARSIFYFAST (G, r, \mathcal{T}, c, C').

Input: hypergraph $G = (E, V)$ with rank r , terminal set \mathcal{T} , threshold parameter c , constant C' .

Output: a (\mathcal{T}, c) -sparsifier H .

- 1 $H \leftarrow G$.
- 2 $iter \leftarrow 0$. /* Number of iterations of the following while-loop. */
- 3 **do**
- 4 $G \leftarrow H$
- 5 $\phi^{-1} \leftarrow 4C'rc^4 \log^3 n$.
- 6 $\{V_i\}_{i=1}^t \leftarrow \text{EXPANDERDECOMPOSE}(G, \phi)$.
- 7 $G' \leftarrow G$ /* Anchored sub-hypergraphs will be separated from G' one by one. */
- 8 **for** each $i = 1, 2, \dots, t$ **do**
- 9 Apply the divide step in Algorithm 1 to G' with terminal \mathcal{T} and bipartition $(V_i, \bigcup_{\ell=i+1}^t V_\ell)$, and get $G_i \leftarrow \hat{G}'[V_i]$ and $G' \leftarrow \hat{G}'[\bigcup_{\ell=i+1}^t V_\ell]$.
- 10 (For each boundary hyperedge e with anchor vertices $v_{e,3}$ and $v_{e,4}$ created on the $\bigcup_{\ell=i+1}^t V_\ell$ side, we assign both $v_{e,3}$ and $v_{e,4}$ to an arbitrary V_j such that $j > i$ and $e \cap V_j \neq \emptyset$.)
- 11 $\{H_i\}_{i=1}^t \leftarrow \{\phi\text{-SPARSIFY}(G_i, V_i \cap \mathcal{T}, \phi, r, c)\}_{i=1}^t$. /* The conquer step. */
- 12 $H \leftarrow H_t$ /* Each sparsifier H_i will be merged with H one by one. */
- 13 **for** each $i = t-1, \dots, 1$ **do**
- 14 Apply the combine step in Algorithm 1 to merge H_i with H . That is, all anchor vertices introduced at the divide step are removed and all separated hyperedges are replaced by the boundary hyperedges before separating V_i from $\bigcup_{\ell=i}^t V_\ell$.
- 15 $iter = iter + 1$.
- 16 **while** $iter < \log m$.
- 17 **return** H .

With EXPANDERDECOMPOSE and ϕ -SPARSIFY procedures introduced above, we are able to construct the (\mathcal{T}, c) -sparsifier on general hypergraphs of size $O(|\mathcal{T}|c^3)$ efficiently. Our algorithm (Algorithm 6) is based on Chalermsook et al. [4] and consists of iterations of EXPANDERDECOMPOSE and ϕ -SPARSIFY. Each iteration implements the divide-and-conquer framework shown by Algorithm 1: we first apply EXPANDERDECOMPOSE and decompose the hypergraph into ϕ -expanders. Then we apply ϕ -SPARSIFY to sparsify the ϕ -expanders. Finally, we glue all sparsifiers of the ϕ -expanders by recovering the inter-cluster hyperedges between the ϕ -expanders. Similar to [4], we prove that $O(\log m)$ iterations suffice to obtain a (\mathcal{T}, c) -sparsifier of $O(|\mathcal{T}|c^3)$ hyperedges.

Due to the page limit, we refer the readers to the full version of this paper to see the details of the algorithms in this section.

References

- 1 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.
- 2 Nikhil Bansal, Ola Svensson, and Luca Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 910–928. IEEE, 2019.
- 3 András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.
- 4 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.
- 5 Chandra Chekuri and Chao Xu. Minimum cuts and sparsification in hypergraphs. *SIAM Journal on Computing*, 47(6):2118–2156, 2018.
- 6 Yu Chen, Sanjeev Khanna, and Ansh Nagda. Near-linear size hypergraph cut sparsifiers. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72. IEEE, 2020.
- 7 David Eppstein, Zvi Galil, Giuseppe F Italiano, and Amnon Nissenzweig. Sparsification—a technique for speeding up dynamic graph algorithms. *Journal of the ACM (JACM)*, 44(5):669–696, 1997.
- 8 Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.
- 9 Wenyu Jin and Xiaorui Sun. Fully dynamic st edge connectivity in subpolynomial time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 861–872. IEEE, 2022.
- 10 Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 598–611, 2021.
- 11 Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. Spectral hypergraph sparsifiers of nearly linear size. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1159–1170. IEEE, 2022.
- 12 Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Information Processing Letters*, 114(7):365–371, 2014.
- 13 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 367–376, 2015.
- 14 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.

- 15 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.
- 16 Yang P. Liu. Vertex sparsification for edge connectivity in polynomial time. *CoRR*, abs/2011.15101, 2020. [arXiv:2011.15101](https://arxiv.org/abs/2011.15101).
- 17 Daniel Lokshтанov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min k -cut. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, 2020.
- 18 Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. *CoRR*, abs/2205.03930, 2022.
- 19 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparsek-connected spanning subgraph of ak -connected graph. *Algorithmica*, 7(1):583–596, 1992.
- 20 Richard Peng, Bryce Sandlund, and Daniel D Sleator. Optimal offline dynamic 2, 3-edge/vertex connectivity. In *Workshop on Algorithms and Data Structures*, pages 553–565. Springer, 2019.
- 21 Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2570–2581. SIAM, 2019.
- 22 Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

Approximation Algorithms for Round-UFP and Round-SAP

Debajyoti Kar ✉

Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Arindam Khan ✉ 🏠 

Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Andreas Wiese ✉ 🏠

Department of Mathematics, Technische Universität München, Germany

Abstract

We study ROUND-UFP and ROUND-SAP, two generalizations of the classical BIN PACKING problem that correspond to the unsplittable flow problem on a path (UFP) and the storage allocation problem (SAP), respectively. We are given a path with capacities on its edges and a set of jobs where for each job we are given a demand and a subpath. In ROUND-UFP, the goal is to find a packing of all jobs into a minimum number of copies (rounds) of the given path such that for each copy, the total demand of jobs on any edge does not exceed the capacity of the respective edge. In ROUND-SAP, the jobs are considered to be rectangles and the goal is to find a non-overlapping packing of these rectangles into a minimum number of rounds such that all rectangles lie completely below the capacity profile of the edges.

We show that in contrast to BIN PACKING, both problems do not admit an asymptotic polynomial-time approximation scheme (APTAS), even when all edge capacities are equal. However, for this setting, we obtain asymptotic $(2 + \varepsilon)$ -approximations for both problems. For the general case, we obtain an $O(\log \log n)$ -approximation algorithm and an $O(\log \log \frac{1}{\delta})$ -approximation under $(1 + \delta)$ -resource augmentation for both problems. For the intermediate setting of the *no bottleneck assumption* (i.e., the maximum job demand is at most the minimum edge capacity), we obtain an absolute 12- and an asymptotic $(16 + \varepsilon)$ -approximation algorithm for ROUND-UFP and ROUND-SAP, respectively.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation Algorithms, Scheduling, Rectangle Packing

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.71

Related Version *Full Version:* <https://arxiv.org/abs/2202.03492>

Funding *Arindam Khan:* Arindam Khan was supported in part by Pratiksha Trust Young Investigator Award, Google CSExplore Award, and Google India Research Award.

Acknowledgements We thank Waldo Gálvez, Afrouz Jabal Ameli, Siba Smarak Panigrahi, and Arka Ray for helpful initial discussions.

1 Introduction

The unsplittable flow on a path problem (UFP) and the storage allocation problem (SAP) are two well-studied problems in combinatorial optimization. In this paper, we study ROUND-UFP and ROUND-SAP, which are two related natural problems that also generalize the classical BIN PACKING problem.

In both ROUND-UFP and ROUND-SAP, we are given as input a path $G = (V, E)$ and a set of n jobs J . We assume that $\{v_0, v_1, \dots, v_m\}$ are the vertices in V from left to right and then for each $i \in \{1, \dots, m\}$ there is an edge $e_i := \{v_{i-1}, v_i\}$. Each job $j \in J$ has integral demand $d_j \in \mathbb{N}$, a source $v_{s_j} \in V$, and a sink $v_{t_j} \in V$. We say that each job j spans the path P_j which we define to be the path between v_{s_j} and v_{t_j} . For every edge $e \in E$, we are given



© Debajyoti Kar, Arindam Khan, and Andreas Wiese;
licensed under Creative Commons License CC-BY 4.0

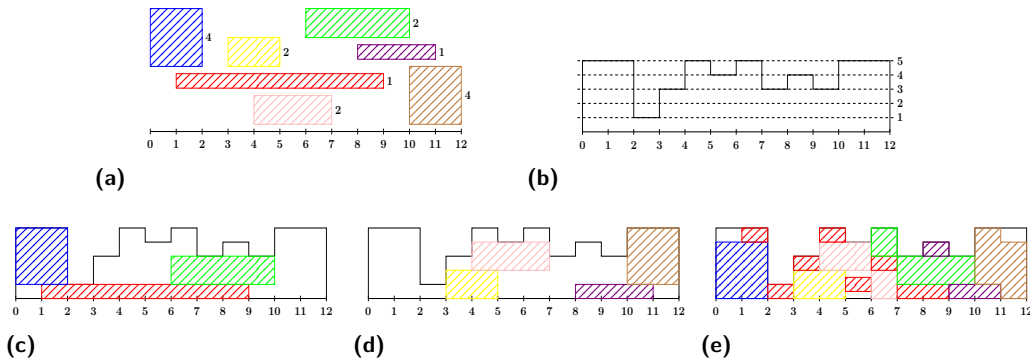
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 71; pp. 71:1–71:19



Leibniz International Proceedings in Informatics

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



■ **Figure 1** (a) A set of 7 jobs with demands written beside; (b) The capacity profile; (c), (d) Any valid ROUND-SAP packing requires at least 2 rounds; (e) A valid ROUND-UFP packing using only 1 round.

an integral capacity c_e . A useful geometric interpretation of the input path and the edge capacities is the following (see Figure 1): consider the interval $[0, m)$ on the x -axis and a function $c: [0, m) \rightarrow \mathbb{N}$. Each edge e_k corresponds to the interval $[k - 1, k)$ and each vertex v_i corresponds to the point i . For edge e_k , we define $c(x) = c_{e_k}$ for each $x \in [k - 1, k)$.

In the ROUND-UFP problem, the objective is to partition the jobs J into a minimum number of sets J_1, \dots, J_k (that we will denote by *rounds*) such that the jobs in each set J_i form a *valid packing*, i.e., they obey the edge capacities, meaning that $\sum_{j \in J_i: e \in P_j} d_j \leq c_e$ for each $e \in E$. In the ROUND-SAP problem, we require to compute additionally for each set J_i a non-overlapping set of rectangles underneath the capacity profile, corresponding to the jobs in J_i (see Figure 1). Formally, we require for each job $j \in J_i$ to determine a height h_j with $h_j + d_j \leq c_e$ for each edge $e \in P_j$, yielding a rectangle $R_j = (s_j, h_j) \times (t_j, h_j + d_j)$, such that for any two jobs $j, j' \in J_i$ we have that $R_j \cap R_{j'} = \emptyset$. Again, the objective is to minimize the number of rounds.

Note that unlike ROUND-SAP, in ROUND-UFP we do not need to pack the jobs as contiguous rectangles. Hence, intuitively in ROUND-UFP we can slice the rectangles vertically and place different slices at different heights. See Figure 1.

ROUND-UFP and ROUND-SAP arise naturally in the setting of resource allocation with connections to many fundamental optimization problems. One possible interpretation of the input path is that each vertex is a computer and that these computers are connected via some wired connection, e.g., optical fiber, ethernet, etc. Each direct connection between two computers is modeled by an edge in the input and it has a certain bandwidth, modeled by the capacity of the edge. Also, certain pairs of computers seek to send data from one to the other, and these pairs are represented by the input jobs. The size of each job corresponds to the amount of data that needs to be transported. Now it can happen that the bandwidths do not support all the jobs. In this case, a remedy is to partition the jobs into groups such that each group is supported by the bandwidths. It is desirable to have as few groups as possible, in order to send the data as quickly as possible. Now each of the mentioned groups corresponds to a round in ROUND-UFP, and therefore we seek to minimize the number of rounds. In ROUND-SAP, we require additionally that each job gets a contiguous portion of the communication bandwidths on each edge and also that this portion is the same on each edge. This is a requirement that arises in practice in wavelength division multiplexing and optical fiber minimization, see [4, 51].

An alternative interpretation of the path is that it represents a time axis. For example, the first edge e_1 would correspond to the time interval $[0, 1)$, the second edge e_2 to the interval $[1, 2)$, etc. The capacities of the edges represent the available amount of some resource of a machine, e.g., memory or transmission frequencies, etc., which might change over time. Each input job then represents a job that needs to be executed during some time interval, i.e., the time interval that corresponds to the edges of the path of the job. Similar to the above, it might be the case that not all jobs can be executed on the same machine because not enough resources are available on a machine. Then, we want to distribute the jobs on several machines, such that all jobs assigned to the same machine can run on that machine, i.e., enough resources are available. Here we assume that each machine has its own resources, e.g., its own CPUs or memory. The jobs on the same machine then form a round according to our definition of ROUND-UFP. Naturally, we seek to minimize the number of rounds, which corresponds to minimizing the number of machines. Similarly as above, in ROUND-SAP, we additionally require that each job gets a contiguous portion of the resource, e.g., a contiguous portion of the frequency spectrum or the memory [11]. Another application is ad-placement, where each job is an advertisement that requires a contiguous portion of the banner [45].

ROUND-UFP and ROUND-SAP are APX-hard as they contain the classical BIN PACKING problem as a special case when G has only one single edge. However, while for BIN PACKING there exists an asymptotic polynomial time approximation scheme (APTAS),¹ it is open whether such an algorithm exists for ROUND-UFP or ROUND-SAP. The best known approximation algorithm for ROUND-UFP is a $O(\min\{\log n, \log m, \log \log c_{\max}\})$ -approximation [35]. For the special case of ROUND-UFP of uniform edge capacities, Pal [49] gave a 3-approximation. Elbassioni et al. [21] gave a 24-approximation algorithm for the problem under the no-bottleneck assumption (NBA) which states that the maximum job demand is upper-bounded by the minimum edge capacity. A result in (the full version of) [45] states that any solution to an instance of UFP can be partitioned into at most 80 sets of jobs such that each of them is a solution to the corresponding SAP instance. This immediately yields approximation algorithms for ROUND-SAP: a 240-approximation for the case of uniform capacities, a 1920-approximation under the NBA, and a $O(\min\{\log n, \log m, \log \log c_{\max}\})$ -approximation for the general case. These are the best known results for ROUND-SAP.

1.1 Our Contributions

First, we show that both ROUND-SAP and ROUND-UFP, unlike the classical BIN PACKING problem, do not admit an APTAS, even in the uniform capacity case. We achieve this via a gap preserving reduction from the 3D matching problem. We create a numeric version of the problem and define a set of hard instances for both ROUND-SAP and ROUND-UFP. Together with a result of Chlebik and Chlebikova [15], we derive an explicit lower bound on the asymptotic approximation ratio for both problems. Our hardness result holds even for the case in which in the optimal packing no round contains more than $O(1)$ jobs, i.e., a case in which we can even enumerate all possible packings in polynomial time.

For the case of uniform edge capacities, we give asymptotic $(2 + \varepsilon)$ -approximation algorithms for both ROUND-UFP and ROUND-SAP, and absolute $(2.5 + \varepsilon)$ and 3-approximation algorithms for the two problems, respectively. This improves upon the previous absolute 3- and 240-approximation algorithms mentioned above. Note that for both problems our factor of 2 is a natural threshold: in many algorithms for UFP and SAP [10, 3, 33, 34, 45, 46], the

¹ For basic definitions related to approximation algorithms, we refer to Section 2.

input jobs are partitioned into jobs that are relatively small and relatively large (compared to the edge capacities). Then, both sets are handled separately with very different sets of techniques. This inherently loses a factor of 2. Our algorithms are based on a connection of our problems to the dynamic storage allocation (DSA) problem and we show how to use some known deep results for DSA [11] in our setting. In DSA the goal is to place some given jobs as non-overlapping rectangles, minimizing the height of the resulting packing. This problem might seem totally unrelated at first glance, in particular, unrelated to ROUND-UFP. The mentioned result produces an alignment of rectangles that correspond to our input jobs, based on which we assign jobs to the different rounds in ROUND-SAP and ROUND-UFP. In one of the two cases that we distinguish, we additionally use a dynamic program that finds an optimal solution for the relatively large jobs in the input. In comparison, the previously best-known algorithm for ROUND-UFP is just a greedy routine that sorts the input jobs and then assigns them greedily into the rounds. The previously best result for ROUND-SAP takes this packing (for ROUND-UFP) and applies a reduction from [45] as a black-box, using a (quite large) factor of 80, yielding a 240-approximation algorithm. In particular, our algorithm for ROUND-SAP yields a much smaller approximation ratio and uses techniques that are much better tailored to ROUND-SAP.

For the general cases of ROUND-UFP and ROUND-SAP, we give an $O(\log \log \min\{m, n\})$ -approximation algorithm. Depending on the concrete values of n , m , and c_{\max} , this constitutes an up to exponential improvement compared to the best known result for ROUND-UFP [35] and for ROUND-SAP (by the reasoning via [45] above). As done previously in the literature, we represent these large jobs by rectangles that are drawn as high as possible underneath the capacity profile and we seek a solution such that in each round, the rectangles corresponding to the jobs in the round are pairwise non-overlapping. In contrast to prior work, we formulate this problem as a configuration-LP and we show that we can solve it in polynomial time via a suitable separation oracle. The integral parts of its solution immediately yield an assignment of some input jobs into at most OPT many rounds. With an additional step, we show that the remaining problem (corresponding to the fractional part of the solution of the configuration-LP) can be reduced to several instances of ROUND-UFP/ ROUND-SAP with the property that each point is overlapped by at most $O(\log m)$ rectangles. This is crucial to be able to apply another novel step: we employ a recent result by Chalermsook and Walczak [12] that yields a bound on the coloring number of rectangle intersection graphs, and we put the rectangles from each color class into a separate round. Since each point is overlapped by at most $O(\log m)$ rectangles, this yields an approximation ratio of $O(\log \log m)$ only, and since w.l.o.g. $m \leq 2n$, also of $O(\log \log n)$. In comparison, the previously best-known algorithm for the general case of ROUND-UFP also uses the viewpoint of the rectangles above, but it pays up to a factor of $O(\log m)$ in order to reduce the general case to the special case in which each rectangle is stabbed by one or two lines from a suitably defined set of lines. For the latter case, it employs a polynomial-time 3-approximation algorithm. In particular, this algorithm neither employs a configuration-LP nor the mentioned result in [12].

Then we study the setting of resource augmentation, i.e., where we can increase the edge capacities by a factor of $1 + \delta$, while still comparing our solution with the optimal solution with original capacities. In this case, we show that we can reduce the given problem to the setting in which the edge capacities are in the range $[1, 1/\delta)$. Applying the algorithm from [35] then yields a $O(\log \log \frac{1}{\delta})$ -approximation for this case for ROUND-UFP, and with a similar argumentation as before also for ROUND-SAP.

Furthermore, for the case of the NBA we improve the absolute approximation ratio from 24 to 12 for ROUND-UFP, and from 1920 to 17 for ROUND-SAP, and we even obtain an asymptotic $(16 + \varepsilon)$ -approximation for ROUND-SAP. For ROUND-SAP we give a black-box

■ **Table 1** Overview of our results. We distinguish the settings according to uniform edge capacities, the no-bottleneck-assumption (NBA), general edge capacities, and general edge capacities with $(1 + \delta)$ -resource augmentation (r.a.). Also, we distinguish between absolute approximation ratios and asymptotic approximation ratios. All listed previous results are absolute approximation ratios.

Problem	Edge capacities	Previous approximation	Improved approximation
ROUND-UFP	uniform	3 [49]	asyp. $2 + \varepsilon$, abs. $2.5 + \varepsilon$
ROUND-SAP	uniform	240 [49, 45]	asyp. $2 + \varepsilon$, abs. 3
ROUND-UFP	NBA	24 [21]	abs. 12
ROUND-SAP	NBA	1920 [21, 45]	asyp. $16 + \varepsilon$, abs. 17
ROUND-UFP	general	$O(\log \min\{n, m, \log c_{\max}\})$ [35]	abs. $O(\log \log \min\{n, m\})$
ROUND-SAP	general	$O(\log \min\{n, m, \log c_{\max}\})$ [35]	abs. $O(\log \log \min\{n, m\})$
ROUND-UFP	general with r.a.	$O(\log \min\{n, m, \log c_{\max}\})$ [35]	abs. $O(\log \log(1/\delta))$
ROUND-SAP	general with r.a.	$O(\log \min\{n, m, \log c_{\max}\})$ [35]	abs. $O(\log \log(1/\delta))$
ROUND-TREE	uniform	6 [23]	asyp. 5.1, abs. 5.5
ROUND-TREE	NBA	64 [21]	asyp. 49, abs. 55

reduction to the case of uniform edge capacities. We first round the job sizes and the edge capacities to powers of 2. Then we partition the input jobs into classes according to their *bottleneck capacities* (the bottleneck capacity of a job is the minimum capacity of an edge through which the job passes). For each class of bottleneck capacities, we define a corresponding instance of ROUND-SAP with uniform edge capacities. The number of different job classes can be super-constant. However, we show that we can combine the solutions for this super-constant number of classes to a global solution for the given instance so that overall we obtain a constant asymptotic approximation ratio of $16 + \varepsilon$ for ROUND-SAP. We remark that our black-box reduction loses a fixed factor of 8, so any improvements for the case of uniform edge capacities would yield an improved approximation ratio for the case of the NBA as well. In comparison, the previously best-known result for ROUND-SAP under the NBA uses the known 24-approximation algorithm for ROUND-UFP and then invokes the mentioned result in [45] as a black-box, yielding a quite large approximation ratio of 1920.

For the setting of round-UFP under the NBA, the previously best-known 24-approximation algorithm in [21] loses a large factor of 16 for packing the small jobs. This algorithm is a simple greedy routine that sorts jobs by their left endpoints and places them appropriately into rounds. In contrast, we first scale up the job demands to integral powers of $1/2$ (assuming the minimum edge capacity to be 1) and then consider jobs based on their bottleneck capacities. We then devise a novel way of classifying the jobs based on the *density* of the jobs on various edges: for one class of jobs we show that a simple greedy algorithm loses only a factor of 4; for the other, our classification scheme ensures that the *congestion* on *every* edge is small which enables us to apply a result in [48], again losing a factor of 4. Also for the large jobs, we improve the best-known 8-approximation to a 4-approximation, thus getting an 12-approximation overall for ROUND-UFP under the NBA.

If in ROUND-UFP we are given a tree instead of a path, we obtain the ROUND-TREE problem. The best known result for it under the NBA is a 64-approximation [21] and a 6-approximation is known for uniform edge capacities [23]. We improve the best known approximation ratio under the NBA to 55 and also provide a 5.5-approximation algorithm for the case of uniform edge capacities.

See Table 1 for an overview of our results.

1.2 Other Related Work

Without the NBA, Epstein et al. [22] showed that no deterministic online algorithm can achieve a competitive ratio better than $\Omega(\log \log n)$ or $\Omega(\log \log \log(c_{\max}/c_{\min}))$ for ROUND-UFP. They also gave a $O(\log c_{\max})$ -competitive algorithm, where c_{\max} is the largest edge capacity. Without the NBA, recently Jahanjou et al. [35] gave a $O(\min(\log m, \log \log c_{\max}))$ -competitive algorithm. For the special case of ROUND-UFP for uniform edge capacities in the online setting, Adamy and Erlebach [2] gave a 195-competitive algorithm, which was subsequently improved to 10 [47, 5].

ROUND-UFP and ROUND-SAP are related to many fundamental optimization problems. For example, ROUND-SAP can be interpreted as an intermediate problem between two-dimensional bin packing (2BP) and the rectangle coloring problem (RC). In 2BP, the goal is to find an axis-parallel nonoverlapping packing of a given set of rectangles (which we can translate in both dimensions) into minimum number of unit square bins. If all edges have the same capacity then ROUND-SAP can be seen as a variant of 2BP in which the horizontal coordinate of each item is fixed and we can choose only the vertical coordinate. For 2BP, the present best asymptotic approximation guarantee is 1.406 [8]. On the other hand, in RC, all rectangles are fixed and the goal is to color the rectangles using a minimum number of colors such that no two rectangles of the same color intersect. For RC, recently Chalermsook and Walczak [12] have given a polynomial-time algorithm that uses only $O(\omega \log \omega)$ colors, where ω is the clique number of the corresponding intersection graph (and hence a lower bound on the number of needed colors). Another related problem is Dynamic Storage Allocation (DSA), where the objective is to pack the given jobs (with fixed horizontal location) such that the maximum vertical height, $\max_j(h_j + d_j)$ (called the *makespan*) is minimized. The current best known approximation ratio for DSA is $(2 + \epsilon)$ [11].

In a sense ROUND-UFP and ROUND-SAP are ‘BIN PACKING-type’ problems, and their corresponding ‘KNAPSACK-type’ problems are UFP and SAP, respectively, where each job has an associated profit and the goal is to select a subset of jobs which can be packed into one single round satisfying the corresponding valid packing constraints. There is a series of work [33, 7, 9, 3, 34, 32] in UFP, culminating in a PTAS [31]. It is maybe surprising that ROUND-UFP does not admit an APTAS, even though UFP admits a PTAS. For SAP, the currently best polynomial time approximation ratio is $2 + \epsilon$ [45], which has been recently improved to $1.969 + \epsilon$ [46] for the case of uniform capacities, and also a quasi-polynomial time $(1.997 + \epsilon)$ -approximation is known for quasi-polynomially bounded input data.

There are many other related problems, such as two-dimensional knapsack [26, 37, 28, 36], strip packing [27, 25, 20, 39], maximum independent set of rectangles [1, 12, 44, 29], guillotine separability of rectangles [41, 40, 42], weighted bipartite edge coloring [43], maximum edge disjoint paths [18], etc. We refer the readers to [38, 17] for an overview of these problems.

2 Preliminaries

Let OPT_{UFP} and OPT_{SAP} denote the optimal number of rounds required to pack all jobs of a given instance of ROUND-UFP and ROUND-SAP, respectively. By simple preprocessing, we can assume that each vertex in V corresponds to endpoint(s) of some job(s) in J , and hence $m \leq 2n - 1$. Job j is said to *pass through* edge e if $e \in P_j$. The *load* on edge e is defined as $l_e := \sum_{e \in P_j} d_j$, the total sum of demands of all jobs passing through e . Let $L := \max_e l_e$ denote the maximum *load*.

We now define some notions related to approximation algorithms. Consider a minimization problem Π . An algorithm \mathcal{A} has approximation guarantee of α ($\alpha > 1$), if $\mathcal{A}(I) \leq \alpha OPT(I)$ for all input instances I of Π . This is also known as *absolute approximation guarantee*. As

common in bin packing literature, we also study asymptotic approximation which intuitively is the approximation ratio when OPT tends to infinity. An algorithm \mathcal{A} has asymptotic approximation guarantee of α , if $\mathcal{A}(I) \leq \alpha OPT(I) + o(OPT(I))$ for all input instances I of Π . A problem admits polynomial time approximation scheme or PTAS (resp. asymptotic polynomial time approximation scheme or APTAS) if for every constant $\varepsilon > 0$, there exists a $(1 + \varepsilon)$ -approximation (resp. $(1 + \varepsilon)$ -asymptotic approximation) algorithm with running time $O(n^{f(1/\varepsilon)})$, for any function f that depends only on ε .

3 Lower Bounds

A simple reduction from the PARTITION problem shows that it is NP-hard to obtain a better approximation ratio than $3/2$ for the classical BIN PACKING problem. However, in the resulting instances, the optimal solutions use only two or three bins. On the other hand, BIN PACKING admits an APTAS [19] and thus, for any $\varepsilon > 0$, a $(1 + \varepsilon)$ -approximation algorithm for instances in which OPT is sufficiently large. Since ROUND-SAP and ROUND-UFP are generalizations of BIN PACKING (even if G has only a single edge), the lower bound of $3/2$ continues to hold. However, maybe surprisingly, we show that unlike BIN PACKING, ROUND-SAP and ROUND-UFP do not admit APTASes, even in the case of uniform edge capacities. More precisely, we provide a lower bound of $(1 + 1/1398)$ on the asymptotic approximation ratio for ROUND-SAP and ROUND-UFP via a reduction from the 2-BOUNDED OCCURRENCE MAXIMUM 3-DIMENSIONAL MATCHING (2-B-3-DM) problem.

In 2-B-3-DM, we are given as input three pairwise disjoint sets $X := \{x_1, x_2, \dots, x_q\}$, $Y := \{y_1, y_2, \dots, y_q\}$, and $Z := \{z_1, z_2, \dots, z_q\}$ and a set of triplets $\mathcal{T} \subseteq X \times Y \times Z$ such that each element of $X \cup Y \cup Z$ occurs in exactly two triplets in \mathcal{T} . Note that $|X| = |Y| = |Z| = q$ and $|\mathcal{T}| = 2q$. A *matching* is a subset $M \subseteq \mathcal{T}$ such that no two triplets in M agree in any coordinate. The goal is to find a matching of maximum cardinality (denoted by OPT_{3DM}). Chlebik and Chlebikova [15] gave the following hardness result.

► **Theorem 1** ([15]). *For 2-B-3-DM there exists a family of instances such that for each instance K of the family, either $OPT_{3DM}(K) < \alpha(q) := \lfloor 0.9690082645q \rfloor$ or $OPT_{3DM}(K) \geq \beta(q) := \lceil 0.979338843q \rceil$, and it is NP-hard to distinguish these two cases.*

Hardness of 2-B-3-DM has been useful in inapproximability results for various (multidimensional) packing, covering, and scheduling problems, e.g. vector packing [52], geometric bin packing [6], geometric bin covering [16], generalized assignment problem [13], etc. Similar to these results, we also use gadgets based on a reduction from 2-B-3-DM to the 4-Partition problem. However, the previous techniques are not directly transferable to our problem due to the inherent differences between these problems. Therefore, we first use the technique from [52] to associate certain integers with the elements of $X \cup Y \cup Z$ and \mathcal{T} and then adapt the numeric data in a different way to obtain the hard instances.

Let $\rho = 32q$ and let \mathcal{V} be the set of $5q$ integers defined as follows: $x'_i = i\rho + 1$, for $1 \leq i \leq |X|$, $y'_j = j\rho^2 + 2$, for $1 \leq j \leq |Y|$, $z'_k = k\rho^3 + 4$, for $1 \leq k \leq |Z|$, $\tau'_l = \rho^4 - k\rho^3 - j\rho^2 - i\rho + 8$, for each triplet $\tau_l = (x_i, y_j, z_k) \in \mathcal{T}$. Define $\gamma = \rho^4 + 15$. The following result is due to Woeginger [52]².

² There was a minor bug in [52], which was fixed by Ray [50]. See Lemma 2 in [50] for the proof of the lemma.

► **Lemma 2** ([52]). *Four integers in \mathcal{V} sum up to the value γ if and only if (i) one of them corresponds to some element $x_i \in X$, one to some element $y_j \in Y$, one to an element $z_k \in Z$, and one to some triplet $\tau_l \in \mathcal{T}$, and if (ii) $\tau_l = (x_i, y_j, z_k)$ holds for these four elements.*

Now, we create a hard instance, tailor-made for our problems. We define that our path $G = (V, E)$ has 40000γ vertices that we identify with the numbers $0, 1, \dots, 40000\gamma$. For each $x_i \in X$ (respectively $y_j \in Y, z_k \in Z$), we specify two jobs $a_{X,i}$ and $a'_{X,i}$ (respectively $a_{Y,j}, a'_{Y,j}$ and $a_{Z,k}, a'_{Z,k}$), which will be called *peers* of each other. Each job j is specified by a triplet (s_j, t_j, d_j) . We define

- $a_{X,i} = (0, 20000\gamma - 4x'_i, 999\gamma + 4x'_i)$ and $a'_{X,i} = (20000\gamma - 4x'_i, 40000\gamma, 1001\gamma - 4x'_i)$,
- $a_{Y,j} = (0, 20000\gamma - 4y'_j, 999\gamma + 4y'_j)$ and $a'_{Y,j} = (20000\gamma - 4y'_j, 40000\gamma, 1001\gamma - 4y'_j)$, and
- $a_{Z,k} = (0, 20000\gamma - 4z'_k, 999\gamma + 4z'_k)$ and $a'_{Z,k} = (20000\gamma - 4z'_k, 40000\gamma, 1001\gamma - 4z'_k)$.

For each $\tau_l \in \mathcal{T}$, we define two jobs b_l and b'_l (also *peers*) by:

- $b_l = (0, 19001\gamma - 4\tau'_l, 999\gamma + 4\tau'_l)$ and $b'_l = (19001\gamma - 4\tau'_l, 40000\gamma, 1001\gamma - 4\tau'_l)$.

Finally let D be a set of $5q - 4\beta(q)$ *dummy* jobs each specified by $(0, 40000\gamma, 2997\gamma)$. We define that each edge $e \in E$ has a capacity of $c_e := c^* := 4000\gamma$. This completes the reduction.

For any job $j = (s_j, t_j, d_j)$ we define its *width* $w_j := t_j - s_j$.

Let $A_X := \{a_{X,i} \mid 1 \leq i \leq q\}$ and $A'_X := \{a'_{X,i} \mid 1 \leq i \leq q\}$. The sets A_Y, A'_Y, A_Z, A'_Z are defined analogously. Let $A := A_X \cup A_Y \cup A_Z$ and $A' := A'_X \cup A'_Y \cup A'_Z$. Finally let $B := \{b_l \mid 1 \leq l \leq 2q\}$ and $B' := \{b'_l \mid 1 \leq l \leq 2q\}$.

To provide some intuition, we first give an upper bound on the number of jobs that can be packed in a round. All following lemmas, statements, and constructions hold for both ROUND-SAP and ROUND-UFP.

► **Lemma 3.** *In any feasible solution any round can contain at most 8 jobs.*

We say that a round is *nice* if it contains exactly 8 jobs. It turns out that such a round corresponds exactly to one element $\tau_l = (x_i, y_j, z_k) \in \mathcal{T}$. We say that the jobs $a_{X,i}, a'_{X,i}, a_{Y,j}, a'_{Y,j}, a_{Z,k}, a'_{Z,k}, b_l$, and b'_l correspond to $\tau_l = (x_i, y_j, z_k)$.

► **Lemma 4.** *We have that a round is nice if and only if there is an element $\tau_l = (x_i, y_j, z_k) \in \mathcal{T}$ such that the round contains exactly the jobs that correspond to $\tau_l = (x_i, y_j, z_k)$.*

Given an optimal solution OPT_{3DM} to 2-B-3-DM with $|OPT_{3DM}| \geq \beta(q)$, we construct a solution as follows:

1. Let \mathcal{M} be any subset of OPT_{3DM} with $|\mathcal{M}| = \beta(q)$. Create $\beta(q)$ nice rounds corresponding to the elements in \mathcal{M} , i.e., for each element $\tau_l = (x_i, y_j, z_k) \in \mathcal{M}$, create a round containing the jobs that correspond to $\tau_l = (x_i, y_j, z_k)$.
2. For each $\tau_l \in \mathcal{T} \setminus \mathcal{M}$, create a round containing b_l and b'_l along with a dummy job.
3. For each $x_i \in X$ (respectively $y_j \in Y, z_k \in Z$) not covered by \mathcal{M} , pack $a_{X,i}$ and $a'_{X,i}$ (respectively $a_{Y,j}, a'_{Y,j}$ and $a_{Z,k}, a'_{Z,k}$) together with one dummy job in one round.

► **Lemma 5.** *If $|OPT_{3DM}| \geq \beta(q)$ then the constructed solution is feasible and it uses at most $5q - 3\beta(q)$ rounds.*

Proof. One can easily check that all constructed rounds are feasible. In step (1) we construct exactly $\beta(q)$ rounds. In step (2), we construct $|\mathcal{T}| - \beta(q) = 2q - \beta(q)$ rounds, since $|\mathcal{T}| = 2q$. In step (3), we construct $3|\mathcal{T} \setminus \mathcal{M}| = 3q - 3|OPT_{3DM}|$ rounds. Hence, overall we construct at most $5q - 3|OPT_{3DM}| \leq 5q - 3\beta(q)$ rounds. ◀

Conversely, assume that $|OPT_{3DM}| < \alpha(q)$ and that we are given any feasible solution to our constructed instance. We want to show that it uses at least $5q - 3\beta(q) + \frac{1}{7}(\beta(q) - \alpha(q))$ rounds. For this, a key property of our construction is given in the following lemma.

► **Lemma 6.** *If a round contains a dummy job, then it can have at most three jobs: at most one dummy job, at most one job from $A \cup B$, and at most one job from $A' \cup B'$.*

Let n_g denote the number of nice rounds in our solution, n_d the number of rounds with a dummy job, and n_b the number of remaining rounds. Note that each of the latter rounds can contain at most 7 jobs each. Since all jobs in $A \cup A' \cup B \cup B'$ need to be assigned to a round, we have that $8n_g + 7n_b + 2n_d \geq 6q + 2|\mathcal{T}| = 10q$. Since the nice rounds correspond to a matching of the given instance of 2-B-3-DM, we have that $n_g \leq \alpha(q)$. Using this, we lower-bound the number of used rounds in the following lemma.

► **Lemma 7.** *If $|OPT_{3DM}| < \alpha(q)$ then the number of rounds in our solution is $n_d + n_g + n_b \geq (5q - 3\beta(q)) + \frac{1}{7}(\beta(q) - \alpha(q))$.*

Proof. Since $8n_g + 7n_b + 2n_d \geq 6q + 2|\mathcal{T}| = 10q$ and $n_d = 5q - 4\beta(q)$, we obtain that $8n_g + 7n_b \geq 8\beta(q)$. Thus $n_g + n_b \geq \frac{8}{7}\beta(q) - \frac{1}{7}n_g$. Since $n_g \leq \alpha(q)$ the number of rounds is at least $n_d + n_g + n_b \geq 5q - 4\beta(q) + \frac{8}{7}\beta(q) - \frac{1}{7}\alpha(q) = (5q - 3\beta(q)) + \frac{1}{7}(\beta(q) - \alpha(q))$. ◀

Now Lemmas 5 and 7 yield our main theorem.

► **Theorem 8.** *There exists a constant $\delta_0 > 1/1398$, such that it is NP-hard to approximate ROUND-UFP and ROUND-SAP in the case of uniform edge capacities with an asymptotic approximation ratio less than $1 + \delta_0$.*

4 Algorithms for Uniform Capacity Case

In this section, we provide asymptotic $(2 + \varepsilon)$ -approximation for ROUND-SAP and ROUND-UFP for the case of uniform edge capacities.

We distinguish two cases, depending on the value of $d_{\max} := \max_{j \in J} d_j$ compared to L .

4.1 Case 1: $d_{\max} \leq \varepsilon^7 L$

First, we invoke an algorithm from [11] for the dynamic storage allocation (DSA) problem. Recall that in DSA the input consists of a set of jobs like in ROUND-SAP and ROUND-UFP, but without upper bounds of the edge capacities. Instead, we seek to define a height h_j for each job j such that the resulting rectangles for the jobs are non-overlapping and the *makespan* $\max_j(h_j + d_j)$ is minimized. The maximum load L is defined as in our setting.

We invoke the following theorem on our input jobs J with $\delta := \varepsilon$.

► **Theorem 9** ([11]). *Assume that we are given a set of jobs J' such that $d_j \leq \delta^7 L$ for each job $j \in J'$. Then there exists an algorithm that produces a DSA packing of J' with makespan at most $(1 + \kappa\delta)L$, where $\kappa > 0$ is some global constant independent of δ .*

Let ξ denote the makespan of the resulting solution to DSA and let c^* denote the (uniform) edge capacity. For each $h \in \mathbb{R}$, we define the horizontal line $\ell_h := \mathbb{R} \times \{h\}$. A job j is said to be *sliced* by ℓ_h if for the computed packing of the jobs it holds that $h_j < h < h_j + d_j$. Now we will transform this into ROUND-SAP or ROUND-UFP packing.

We define a set of rounds Γ_1 . The set Γ_1 contains a round for each integer i with $0 \leq i \leq \lfloor \xi/c^* \rfloor$ and this round contains all jobs lying between ℓ_{ic^*} and $\ell_{(i+1)c^*}$. In this case, we define *congestion* $r = \lceil L/c^* \rceil$ (clearly, $r \leq OPT_{UFP} \leq OPT_{SAP}$). Thus, $|\Gamma_1| \leq \lfloor (1 + \kappa\varepsilon)L/c^* \rfloor + 1 \leq (1 + \kappa\varepsilon)r + 1$. There are two subcases.

Subcase A: Assume that $r > 1/(2\kappa\varepsilon)$. In this case $|\Gamma_1| \leq (1 + 3\kappa\varepsilon)r$. We define a set of rounds Γ_2 as follows. For each integer i with $1 \leq i \leq \lfloor \xi/c^* \rfloor$, Γ_2 has a round containing all jobs that are sliced by ℓ_{ic^*} . Thus $|\Gamma_2| \leq \lfloor (1 + \kappa\varepsilon)L/c^* \rfloor \leq (1 + \kappa\varepsilon)r$. Hence, the total number of rounds is bounded by $(2 + 4\kappa\varepsilon)r \leq (2 + O(\varepsilon))OPT_{UFP} \leq (2 + O(\varepsilon))OPT_{SAP}$.

71:10 Approximation Algorithms for Round-UFP and Round-SAP

Subcase B: Assume that $r \leq 1/(2\kappa\varepsilon)$. Now $\xi \leq (1+\kappa\varepsilon)L$, and therefore $\xi - L \leq \kappa\varepsilon L \leq c^/2$. Hence, we have $|\Gamma_1| \leq r + 1$ and the $(r + 1)^{\text{th}}$ round is filled up to a capacity of at most $c^*/2$ on each edge. Now the total load of the set of jobs that are sliced by $(\ell_{ic^*})_{1 \leq i \leq r}$ is at most $r \cdot \varepsilon^7 L$. We now invoke the following result on DSA to this set of jobs.*

► **Theorem 10** ([30]). *Let J' be a set of jobs with load L . Then a DSA packing of J' of makespan at most $3L$ can be computed in polynomial time.*

Thus the makespan of the computed solution is at most $3r \cdot \varepsilon^7 L \leq 3 \cdot \frac{1}{2\kappa\varepsilon} \cdot \varepsilon^7 \cdot \frac{c^*}{2\kappa\varepsilon} \leq c^*/2$, if ε is small enough. Hence these jobs can be added to the $(r + 1)^{\text{th}}$ round of Γ_1 . Therefore, we get a packing of J using at most $r + 1 \leq OPT_{UFP} + 1 \leq OPT_{SAP} + 1$ rounds.

4.2 Case 2: $d_{\max} > \varepsilon^7 L$

For this case, we have $c^* \geq d_{\max} > \varepsilon^7 L$ and therefore $r = \lceil L/c^* \rceil \leq 1/\varepsilon^7$. We partition the input jobs into *large* and *small* jobs by defining $J_{\text{large}} := \{j \in J \mid d_j > \varepsilon^{56} L\}$ and $J_{\text{small}} := \{j \in J \mid d_j \leq \varepsilon^{56} L\}$.

We start with the small jobs J_{small} . First, we apply Theorem 9 to them with $\delta := \varepsilon^8$ and obtain a DSA packing \mathcal{P} for them. We transform it into a solution to ROUND-SAP with at most $r + 1$ rounds as follows: we introduce a set Γ_1 consisting of $r + 1$ rounds exactly as in the previous case (when $r \leq 1/(2\kappa\varepsilon)$). The $(r + 1)^{\text{th}}$ round would be filled up to a capacity of at most $\kappa\varepsilon^8 L \leq \kappa\varepsilon c^*$. Again applying Theorem 10 to the remaining jobs, we get a DSA packing of makespan at most $3r \cdot \varepsilon^{56} L \leq 3 \cdot (1/\varepsilon^7) \cdot \varepsilon^{56} \cdot (c^*/\varepsilon^7) \leq c^*/2$, and therefore these jobs can be packed inside the $(r + 1)^{\text{th}}$ round. Hence, there exists a packing of J_{small} using at most $r + 1 \leq OPT_{UFP} + 1 \leq OPT_{SAP} + 1$ rounds.

Now we consider the large jobs J_{large} . Our strategy is to compute an optimal solution for them via dynamic programming (DP). Intuitively, our DP orders the jobs in J_{large} non-decreasingly by their respective source vertices and assigns them to the rounds in this order. Since the jobs are large, each edge is used by at most $1/\varepsilon^{56}$ large jobs, and using interval coloring one can show easily that at most $1/\varepsilon^{56} = O_\varepsilon(1)$ rounds suffice (e.g., we can color the jobs with $1/\varepsilon^{56}$ colors such that no two jobs with intersecting paths have the same color). This already gives a packing into at most $(2 + \varepsilon)OPT_{UFP} + O_\varepsilon(1)$ rounds. However, our DP will return an improved packing with at most $(2 + \varepsilon)OPT_{UFP} + 1$ rounds. In our DP we have a cell for each combination of an edge e and the assignment of all jobs passing through e to the rounds. Given this, the corresponding subproblem is to assign additionally all jobs to the rounds whose paths lie completely on the right of e .

For ROUND-SAP we additionally want to bound the number of possible heights h_j . To this end, we restrict ourselves to packings that are normalized which intuitively means that all jobs are pushed up as much as possible. Formally, we say that a packing for a set of jobs J' inside a round is *normalized* if for every $j \in J'$, either $h_j + d_j = c^*$ or $h_j + d_j = h_{j'}$ for some $j' \in J'$ such that $P_j \cap P_{j'} \neq \emptyset$.

► **Lemma 11.** *Consider a valid packing of a set of jobs $J' \subseteq J_{\text{large}}$ inside one round. Then there is also a packing for J' that is normalized.*

Now the important insight is that in a normalized packing of large jobs, the height h_j of a job j is the difference of (the top height level) c^* and the sum of at most $1/\varepsilon^{56}$ jobs in J_{large} . Thus, the number of possible heights is bounded by $n^{O(1/\varepsilon^{56})}$ and we can compute all these possible heights before starting our DP.

► **Lemma 12.** *Given J_{large} we can compute a set \mathcal{H} of $n^{O(1/\varepsilon^{56})}$ values such that in any normalized packing of a set $J' \subseteq J_{\text{large}}$ inside one round, the height h_j of each job $j \in J'$ is contained in \mathcal{H} .*

Now we can compute the optimal packing via a dynamic program as described above, which yields the following lemma.

► **Lemma 13.** *Consider an instance of ROUND-UFP or ROUND-SAP with a set of jobs J' satisfying the following conditions:*

(i) *The number of jobs using any edge is bounded by ω .*

(ii) *In the case of ROUND-SAP there is a given set \mathcal{H}' of allowed heights for the jobs.*

Then we can compute an optimal solution to the given instance in time $(n|\mathcal{H}'|)^{O(\omega)}$.

We invoke Lemma 13 with $J' := J_{\text{large}}$, $\omega = 1/\varepsilon^{56}$, and in the case of ROUND-SAP we define \mathcal{H}' to be the set \mathcal{H} due to Lemma 12. This yields at most OPT_{UFP} rounds in total for the large jobs J_{large} . Hence, we obtain a packing of J using at most $2 \cdot OPT_{UFP} + 1$ rounds.

Case 1 and 2 together imply a packing into at most $(2 + \varepsilon)OPT_{UFP} + 1$ rounds. This yields our main theorem for the case of uniform edge capacities.

► **Theorem 14.** *For any $\varepsilon > 0$, there exist asymptotic $(2 + \varepsilon)$ -approximation algorithms for ROUND-SAP and ROUND-UFP, assuming uniform edge capacities.*

We now derive some bounds on the absolute approximation ratios. If $OPT_{SAP} = 1$, our algorithm would return a packing using at most $(2 + \varepsilon) \cdot 1 + 1$ rounds, and hence at most 3 rounds. If $OPT_{SAP} \geq 2$, then our algorithm uses at most $(2 + \varepsilon) \cdot OPT_{SAP} + OPT_{SAP}/2 = (2.5 + \varepsilon)OPT_{SAP}$ rounds. Hence, we obtain the following result.

► **Theorem 15.** *There exists a polynomial time 3-approximation algorithm for ROUND-SAP, assuming uniform edge capacities.*

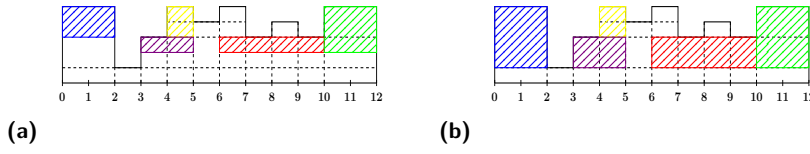
For ROUND-UFP, it is easy to check whether $OPT_{UFP} = 1$ by checking whether $L \leq c^*$. Otherwise, $OPT_{UFP} \geq 2$ and similar as above, the number of rounds used would be at most $(2.5 + \varepsilon) \cdot OPT_{UFP}$. This gives an improvement over the result of Pal [49].

► **Theorem 16.** *For any $\varepsilon > 0$, there exists a polynomial time $(2.5 + \varepsilon)$ -approximation algorithm for ROUND-UFP, assuming uniform edge capacities.*

5 General Case

In this section, we present our algorithms for the general cases of ROUND-UFP and ROUND-SAP. We begin with our $O(\log \log \min\{n, m\})$ -approximation algorithms where we consider ROUND-UFP first and describe later how to extend our algorithm to ROUND-SAP. We split the input jobs into large and small jobs. For each job j , we denote by \mathbf{b}_j the minimum capacity of the edges in P_j , i.e., $\min\{c_e : e \in P_j\}$. We call \mathbf{b}_j the *bottleneck capacity* of the job j . We define $J_{\text{large}} := \{j \in J \mid d_j > \mathbf{b}_j/4\}$ and $J_{\text{small}} := \{j \in J \mid d_j \leq \mathbf{b}_j/4\}$. For the small jobs, we invoke a result by Elbassioni et al. [21] that yields a 16-approximation.

► **Theorem 17 ([21]).** *We are given an instance of ROUND-UFP with a set of jobs J' such that $d_j \leq \frac{1}{4}\mathbf{b}_j$ for each job $j \in J'$. Then there is a polynomial time algorithm that computes a 16-approximate solution to J' .*



■ **Figure 2** (a) The capacity profile along with the sets of lines \mathcal{H} and \mathcal{V} and some top-drawn jobs; (b) The jobs after processing;

Now consider the large jobs J_{large} . For each job $j \in J_{\text{large}}$, we define a rectangle $R_j = (s_j, \mathbf{b}_j - d_j) \times (t_j, \mathbf{b}_j)$. Note that R_j corresponds to the rectangle for j in ROUND-SAP if we assign j the maximum possible height h_j (which is $h_j := \mathbf{b}_j - d_j$). We say that a set of jobs $J' \subseteq J_{\text{large}}$ is *top-drawn* (underneath the capacity profile), if their rectangles are pairwise non-overlapping, i.e., if $R_j \cap R_{j'} = \emptyset$ for any $j, j' \in J'$. If a set of jobs J' is top-drawn, then it clearly forms a feasible round in ROUND-UFP. However, not every feasible round of ROUND-UFP is top-drawn. Nevertheless, we look for a solution to ROUND-UFP in which the jobs in each round are top-drawn. The following lemma implies that this costs only a factor of 8 in our approximation ratio.

► **Lemma 18** ([10]). *Let $J' \subseteq J_{\text{large}}$ be a set of jobs packed in a feasible round for a given instance of ROUND-UFP. Then J' can be partitioned into at most 8 sets such that each of them is top-drawn.*

Let $\mathcal{R}_{\text{large}} := \{R_j | j \in J_{\text{large}}\}$ denote the rectangles corresponding to the large jobs and let ω_{large} be their clique number, i.e., the size of the largest set $\mathcal{R}' \subseteq \mathcal{R}_{\text{large}}$ such that all rectangles in \mathcal{R}' pairwise overlap. As a consequence of Helly's theorem³ for such a set of axis-parallel rectangles \mathcal{R}' there must be a point in which all rectangles in \mathcal{R}' overlap. Note that we need at least ω_{large} rounds since we seek a solution with only top-drawn jobs in each round.

We first reduce the original instance to the case where there are only $O(m^2)$ many distinct job demands. For each $i \in \{1, \dots, m\}$, let p_i denote the point (i, c_{e_i}) . We draw a horizontal and a vertical line segment passing through p_i and lying completely under the capacity profile (see Figure 2(a)). This divides the region underneath the capacity profile into at most m^2 regions. Let \mathcal{H} denote the set of horizontal lines and \mathcal{V} denote the set of vertical lines drawn. Thus, the top edge of any rectangle corresponding to a large job must touch a line in \mathcal{H} . Now consider any rectangle R_j corresponding to a job j . Let $h \in \mathcal{H}$ be the horizontal line segment lying just below the bottom edge of R_j . We increase the value of d_j so that the the bottom edge of R_j now touches the line segment h (see Figure 2(b)). Since the rectangles were top-drawn, the clique number of this new set of large rectangles (denoted by $\mathcal{R}'_{\text{large}}$) does not change. Also any feasible packing of $\mathcal{R}'_{\text{large}}$ is a feasible packing of $\mathcal{R}_{\text{large}}$.

Note that $\mathcal{R}'_{\text{large}}$ contains at most m^4 distinct types of jobs: the endpoints s_j and t_j can be chosen in $\binom{m+1}{2} \leq m^2$ ways and the top and bottom edges of R_j must coincide with two lines from \mathcal{H} , which can be again chosen in $\binom{m}{2} \leq m^2$ ways. Let U denote the number of types of job of the given instance and let $\mathcal{R}'_{\text{large}} = J_1 \cup J_2 \cup \dots \cup J_U$ be the decomposition of $\mathcal{R}'_{\text{large}}$ into the U distinct job types.

³ Project the family of rectangles onto both x and y dimensions. These projections are intervals that are pairwise intersecting. Helly's theorem states that a pairwise intersecting set of intervals share a common point. Then the product of each of these 1 dimensional common intersections is shared by each of the rectangles.

We now formulate the configuration LP for this instance. Let \mathcal{C} denote the set of all possible configurations of a round containing jobs from $\mathcal{R}'_{\text{large}}$, drawn as top-drawn sets. For each $C \in \mathcal{C}$, we introduce a variable x_C , which stands for the number of rounds having configuration $C \in \mathcal{C}$. We write $J_k \triangleleft C$ if configuration C contains a job from J_k (note that C can contain at most one job from J_k). Then the relaxed configuration LP and its dual (which contains a variable y_k for each set J_k) are as follows.

$$\begin{array}{ll} \text{minimize} & \sum_{C \in \mathcal{C}} x_C \\ \text{subject to} & \sum_{C: J_k \triangleleft C} x_C \geq |J_k|, \quad k = 1, \dots, U \\ & x_C \geq 0, \quad C \in \mathcal{C} \end{array} \quad \begin{array}{ll} \text{maximize} & \sum_{k=1}^U |J_k| y_k \\ \text{subject to} & \sum_{k: J_k \triangleleft C} y_k \leq 1, \quad C \in \mathcal{C} \\ & y_k \geq 0, \quad k = 1, \dots, U \end{array}$$

The dual LP can be solved via the ellipsoid method with a suitable separation oracle. We interpret y_k as the weight of each job in J_k . Given $(y_k)_{k \in \{1, \dots, U\}}$, the separation problem asks whether there exists a configuration where jobs are drawn as top-drawn sets and the total weight of all the jobs in the configuration exceeds 1. For this, we invoke the following result of Bonsma et al. [10].

► **Theorem 19** ([10]). *Given an instance of UFP with a set of jobs J' , the maximum-weight top-drawn subset of J' can be computed in $O(nm^3)$ time.*

Let $(x_C^*)_{C \in \mathcal{C}}$ be an optimal basic solution of the primal LP. By the rank lemma, there are at most U configurations C for which x_C^* is non-zero. For each non-zero x_C^* , we introduce $\lfloor x_C^* \rfloor$ rounds with configuration C , thus creating at most $8 \cdot OPT_{UFP}$ rounds (due to Lemma 18). Now let $\mathcal{R}''_{\text{large}} \subseteq \mathcal{R}'_{\text{large}}$ be the large jobs that are yet to be packed and let ω''_{large} be their clique number (and note that $\omega''_{\text{large}} \leq 8 \cdot OPT_{UFP}$). In particular, a feasible solution to the configuration LP for the rectangles in $\mathcal{R}''_{\text{large}}$ is to select one more round for each configuration C with $x_C^* > 0$. Therefore, we conclude that $\omega''_{\text{large}} \leq U \leq m^4$ since there are at most U configurations C with $x_C^* > 0$ and for each point, each configuration contains at most one rectangle covering this point.

Our strategy is to invoke the following theorem on $\mathcal{R}''_{\text{large}}$.

► **Theorem 20** ([12]). *Given a set of rectangles with clique number ω , in polynomial time, we can compute a coloring of the rectangles using $O(\omega \log \omega)$ colors such that no two rectangles of the same color intersect.*

Thus, if $\omega''_{\text{large}} = O(\log m)$ then we obtain an $O(\log \log m)$ -approximation as desired. However, it might be that ω''_{large} is larger. In that case, we partition $\mathcal{R}''_{\text{large}}$ into $\omega''_{\text{large}}/\log m$ sets, such that each of them has a clique size of $O(\log m)$.

► **Lemma 21.** *There is a randomized polynomial time algorithm that w.h.p. computes a partition $\mathcal{R}''_{\text{large}} = \mathcal{R}_1 \dot{\cup} \dots \dot{\cup} \mathcal{R}_{\omega''_{\text{large}}/\log m}$ such that for each set \mathcal{R}_i , the corresponding clique size is at most $O(\log m)$.*

Proof. We split the rectangles $\mathcal{R}''_{\text{large}}$ uniformly at random into $\omega''_{\text{large}}/\log m$ sets $\mathcal{R}_1, \dots, \mathcal{R}_{\omega''_{\text{large}}/\log m}$. Thus the expected clique size in each set \mathcal{R}_i at any point p under the profile is at most $\log m$. Using the Chernoff bound, the probability that the clique size at p is more than $8 \log m$ is at most $2^{-8 \log m} = 1/m^8$. As before, we draw the set of horizontal and vertical lines \mathcal{H} and \mathcal{V} , respectively, under the capacity profile, dividing the region underneath the profile into at most m^2 regions. Clearly, the clique number must

be the same at all points inside any such region. Thus the probability that there exists a point p under the capacity profile where the clique size is more than $8 \log m$ is at most $m^2/m^8 \leq 1/m^6$. Hence using union bound, probability that clique size is more than $8 \log m$ at some point in some set \mathcal{R}_i is at most $1/m^2$ (since $\omega''_{\text{large}} \leq m^4$). \blacktriangleleft

We apply Theorem 20 to each set \mathcal{R}_i separately and thus obtain a coloring with $O(\log m \log \log m)$ colors. Thus, for all sets \mathcal{R}_i together we use at most $\frac{\omega''_{\text{large}}}{\log m} O(\log m \log \log m) = O(\omega''_{\text{large}} \log \log m)$ colors. We pack the jobs from each color class to a separate round for our solution to ROUND-UFP. This yields an $O(\log \log m)$ -approximation, together with Theorem 17. Since $m \leq 2n - 1$ after our preprocessing, our algorithms are also $O(\log \log n)$ -approximation algorithms.

► **Theorem 22.** *There exists a randomized $O(\log \log \min\{n, m\})$ -approximation algorithm for ROUND-UFP for general edge capacities.*

In order to obtain an algorithm for ROUND-SAP, we invoke the following lemma due to [45] to each round of the computed solution to ROUND-UFP.

► **Lemma 23** ([45]). *Let J' be the set of jobs packed in a feasible round for a given instance of ROUND-UFP. Then in polynomial time we can partition J' into $O(1)$ sets and compute a height h_j for each job $j \in J'$ such that each set yields a feasible round of ROUND-SAP.*

This yields a solution to ROUND-SAP with only $O(OPT_{UFP} \log \log m) \leq O(OPT_{SAP} \log \log m)$ many rounds.

► **Theorem 24.** *There exists a randomized $O(\log \log \min\{n, m\})$ -approximation algorithm for ROUND-SAP for general edge capacities.*

5.1 An $O(\log \log \frac{1}{\delta})$ -approximation algorithm with $(1 + \delta)$ -resource augmentation

We show that if we are allowed a resource augmentation of a factor of $1 + \delta$ for some $\delta > 0$, we can get an $O(\log \log \frac{1}{\delta})$ -approximation for both ROUND-SAP and ROUND-UFP.

Consider ROUND-UFP first. Recall that \mathfrak{b}_j denotes the bottleneck capacity of job j , i.e. $\min\{c_e : e \in P_j\}$. For each $i \in \mathbb{N}$ we define the set $J^{(i)} := \{j \in J \mid \mathfrak{b}_j \in [1/\delta^i, 1/\delta^{i+1}]\}$ and consider one resulting set $J^{(i)}$. By definition no job in $J^{(i)}$ uses an edge whose capacity is less than $1/\delta^i$. Also, we can assume that the capacity of each edge is at most $2/\delta^{i+1}$ since for each edge e with a capacity of more than $1/\delta^{i+1}$, each jobs using it must also use the closest edge on the left or on the right of e with a capacity of at most $1/\delta^{i+1}$. Thus, in a feasible round, e is used by jobs from $J^{(i)}$ with a total demand of at most $2/\delta^{i+1}$. Thus, via scaling we can assume that the edge capacities are in the range $[1, 2/\delta)$ if our input consists of $J^{(i)}$ only.

► **Lemma 25.** *Let $J^{(i)} := \{j \in J \mid \mathfrak{b}_j \in [1/\delta^i, 1/\delta^{i+1}]\}$. For packing jobs in $J^{(i)}$, it can be assumed that the capacity of each edge lies in the range $[1/\delta^i, 2/\delta^{i+1})$.*

Hence using the following theorem, we get a $O(\log \log \frac{1}{\delta})$ -approximate solutions for each $J^{(i)}$, which in particular uses at most $O(OPT_{UFP} \log \log \frac{1}{\delta})$ rounds.

► **Theorem 26** ([35]). *There is a polynomial time $O(\log \log \frac{c_{\max}}{c_{\min}})$ -approximation algorithm for ROUND-UFP.*

Next, we argue that we can combine the rounds computed for the sets $J^{(i)}$. More precisely, we show that if we take one round from each set $J^{(0)}, J^{(2)}, J^{(4)}, \dots$ and form their union, then they form a feasible round for the given instance under $(1 + \delta)$ -resource augmentation: as argued above, each set $J^{(i)}$ uses at most a capacity of $2/\delta^{i+1}$ on each edge e in any feasible round. Also, if an edge e is used by a job from a set $J^{(i')}$ for an even $i' \in \mathbb{N}$ in a feasible round, then e has a capacity of at least $1/\delta^{i'}$. Therefore, if we have $(1 + 2\delta)$ -resource augmentation available, then by a geometric sum argument the gained capacity on e is enough for one round of each of the sets $J^{(i'-2)}, J^{(i'-4)}, J^{(i'-6)}, \dots$.

With a similar argumentation we can show that we obtain a feasible solution if we take one round from each set $J^{(1)}, J^{(3)}, J^{(5)}, \dots$.

► **Lemma 27.** *Take one computed round for each set $J^{(2k)}$ with $k \in \mathbb{N}$ or one computed round from each set $J^{(2k+1)}$ with $k \in \mathbb{N}$, and let J' be their union. Then J' is a feasible round for the given instance of ROUND-UFP under $(1 + \delta)$ -resource augmentation.*

Thus, due to Lemma 27 we obtain a solution with at most $O(OPT_{UFP} \log \log \frac{1}{\delta})$ rounds for the overall instance. As earlier, we take the given ROUND-UFP solution and apply Lemma 23 to it, which yields a solution to ROUND-SAP with at most $O(OPT_{SAP} \log \log \frac{1}{\delta})$ rounds.

► **Theorem 28.** *There exists an $O(\log \log \frac{1}{\delta})$ -approximation algorithm for ROUND-SAP and ROUND-UFP for general edge capacities and $(1 + \delta)$ -resource augmentation.*

6 Algorithms for the no-bottleneck-assumption

In this section, we present a $(16 + \varepsilon)$ -approximation algorithm for ROUND-SAP and a 12-approximation for ROUND-UFP, both under the no-bottleneck-assumption (NBA).

6.1 Algorithm for Round-SAP

For our algorithm for ROUND-SAP under NBA, we first scale down all job demands and edge capacities so that $c_{\min} := \min_{e \in E} c_e = 1$ (note that this implies that $d_j \leq 1$ for each job $j \in J$). Then, we scale down all edge capacities to the nearest power of 2 and let $(c'_e)_{e \in E}$ denote the new edge capacities. Define a set of horizontal lines $\mathcal{L} := \{\ell_{2^k} | k \in \mathbb{N}\}$. Let OPT'_{SAP} denote the optimal solution for the rounded down capacities $(c'_e)_{e \in E}$ under the additional constraint that there must be no job whose rectangle intersects a line in \mathcal{L} .

It can be shown that given a valid ROUND-SAP packing \mathcal{P} of a set of jobs J' for the edge capacities $(c_e)_{e \in E}$, there exists a valid packing of J' into 4 rounds $\mathcal{R}^1, \mathcal{R}^2, \mathcal{R}^3, \mathcal{R}^4$, under profile $(c'_e)_{e \in E}$ such that no job is sliced by a line in \mathcal{L} . This implies that $OPT'_{SAP} \leq 4 \cdot OPT_{SAP}$.

We shall now obtain a valid packing of J for the edge capacities $(c'_e)_{e \in E}$. Let $c'_{\max} := \max_{e \in E} c'_e$ and for each $i \in \{0, 1, \dots, \log c'_{\max}\}$ let $J^{(i)} \subseteq J$ denote the set of jobs with bottleneck capacity 2^i according to $(c'_e)_{e \in E}$. For each set $J^{(i)}$ we create a new (artificial) instance with uniform edge capacities: in the instance for $J^{(0)}$ all edges have capacity 1, and for each $i \in \{1, 2, \dots, \log c'_{\max}\}$ in the instance for $J^{(i)}$ all edges have capacity 2^{i-1} . For each $i \in \{0, 1, 2, \dots, \log c'_{\max}\}$, denote by $OPT^{(i)}$ the number of rounds needed in the optimal solution to the instance for $J^{(i)}$. Since in the solution OPT'_{SAP} , no rectangle is intersected by a line in \mathcal{L} , for each set $J^{(i)}$ we can easily rearrange the jobs in $J^{(i)}$ in OPT'_{SAP} such that we obtain a solution for $J^{(i)}$ with at most $2 \cdot OPT'_{SAP}$ rounds.

For each set $J^{(i)}$ we invoke our asymptotic $(2 + \varepsilon)$ -approximation algorithm for ROUND-SAP for uniform edge capacities (see Section 4) and obtain a solution $\Gamma^{(i)}$ which hence uses $\Gamma^{(i)} \leq (2 + \varepsilon) \cdot OPT^{(i)} + 1 \leq (4 + O(\varepsilon)) \cdot OPT'_{SAP} + 1 \leq (16 + O(\varepsilon)) \cdot OPT_{SAP} + 1$ rounds. Finally, we combine the solutions $\Gamma^{(i)}$ for all $i \in \{0, 1, 2, \dots, \log c'_{\max}\}$ to one global solution of J , which uses at most $\max^{(i)} \{\Gamma^{(i)}\}$ many rounds. Hence we have the following theorem.

► **Theorem 29.** *For any $\varepsilon > 0$, there exists a polynomial-time asymptotic $(16 + \varepsilon)$ -approximation and an absolute 17-approximation algorithm for ROUND-SAP under the NBA.*

6.2 Algorithm for Round-UFP

In this section, we present a 12-approximation for ROUND-UFP under NBA. In ROUND-UFP, it is not clear how to bootstrap the algorithm for the uniform case as we did for ROUND-SAP, since in the optimal solution it might not be possible to draw the jobs as non-overlapping rectangles. Instead, our algorithm refines combinatorial properties from [21] to obtain an improved approximation ratio.

For any edge e , we define the *congestion* $r_e := \lceil l_e/c_e \rceil$, and $r = \max_e r_e$ denotes the maximum congestion of any edge of the path. A result in [48] states that for the special case when all jobs have identical demands and all edge capacities are integral multiples of the demand, a ROUND-UFP packing using r rounds can be found efficiently.

Via scaling, we assume that $c_{\min} = 1$ and the demand of each job is at most 1. Let $J_{\text{large}} := \{j \in J \mid d_j > 1/2\}$ and $J_{\text{small}} := J \setminus J_{\text{large}}$. For jobs in J_{large} , applying the result due to [48] after scaling up all the demands to 1 and scaling down the capacity of each edge to the nearest integer directly yields a packing using at most $4r$ rounds.

Now for each job $j \in J_{\text{small}}$, we round up its demand to the next larger power of $1/2$. For each $i \in \mathbb{N}$, let $J^{(i)}$ denote the set of jobs whose demands after rounding equal $\frac{1}{2^i}$. For each edge e and each $i \in \mathbb{N}$, we define $n_{e,i} := |\{j \in J^{(i)} \mid e \in P_j\}|$. We partition each set $J^{(i)}$ into the sets $J'^{(i)} = \{j \in J^{(i)} \mid \exists e \in P_j : n_{e,i} < 2r\}$ and $J''^{(i)} = J \setminus J'^{(i)}$. Let $n'_{e,i} := |\{j \in J'^{(i)} \mid e \in P_j\}|$ and $n''_{e,i} := |\{j \in J''^{(i)} \mid e \in P_j\}|$. Clearly, $n'_{e,i} < 4r$ for each edge e and each i .

Since there are at most $4r$ jobs from each $J'^{(i)}$ over any edge, the jobs in $\bigcup_i J'^{(i)}$ can be easily packed into $4r$ rounds using interval coloring. For the jobs in $\bigcup_i J''^{(i)}$, we partition the available capacity inside each round among the sets $J''^{(i)}$; formally for each edge e , we reserve a capacity of $\frac{1}{2^i} \cdot \lfloor \frac{n_{e,i}}{2r} \rfloor$ for $J''^{(i)}$ (note $\sum_i \frac{1}{2^i} \lfloor \frac{n_{e,i}}{2r} \rfloor \leq c_e$). The resulting congestion of any edge having non-zero capacity is $\frac{\frac{1}{2^i} n''_{e,i}}{\frac{1}{2^i} \lfloor \frac{n_{e,i}}{2r} \rfloor} \leq \frac{\frac{n_{e,i}}{2r}}{\lfloor \frac{n_{e,i}}{2r} \rfloor} \cdot 2r \leq 4r$. Thus using the algorithm due to [48], jobs in $J''^{(i)}$ can be packed into at most $4r$ rounds with these capacities. Hence we obtain a valid packing of J_{small} using at most $4r + 4r = 8r$ rounds.

We thus have the following theorem.

► **Theorem 30.** *There exists a polynomial-time 12-approximation algorithm for ROUND-UFP under the NBA.*

7 Algorithms for Round-Tree

Extending the results for ROUND-UFP, using results on path coloring [24] and multicommodity demand flow [14], we obtain the following results for ROUND-TREE (we refer the reader to the full version of this paper for details).

► **Theorem 31.** *For ROUND-TREE, there exists a polynomial-time asymptotic (resp. absolute) 5.1- (resp. 5.5-) approximation algorithm for uniform edge capacities and an asymptotic (resp. absolute) 49- (resp. 55-) approximation algorithm for the general case under the NBA.*

References

- 1 Anna Adamaszek, Sarel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *Journal of the ACM*, 66(4):29:1–29:40, 2019.
- 2 Udo Adamy and Thomas Erlebach. Online coloring of intervals with bandwidth. In *WAOA*, pages 1–12. Springer, 2003.
- 3 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing $2 + \epsilon$ approximation for unsplittable flow on a path. In *SODA*, pages 26–41, 2014.
- 4 Matthew Andrews and Lisa Zhang. Bounds on fiber minimization in optical networks with fixed fiber capacity. In *24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 409–419, 2005.
- 5 Yossi Azar, Amos Fiat, Meital Levy, and NS Narayanaswamy. An improved algorithm for online coloring of intervals with bandwidth. *Theoretical Computer Science*, 363(1):18–27, 2006.
- 6 Nikhil Bansal, José R. Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: inapproximability results and approximation schemes. *Mathematics of operations research*, 31(1):31–49, 2006.
- 7 Nikhil Bansal, Zachary Friggstad, Rohit Khandekar, and Mohammad R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. *ACM Transactions on Algorithms (TALG)*, 10(1):1–15, 2014.
- 8 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *SODA*, pages 13–25, 2014.
- 9 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.
- 10 Paul S. Bonsma, Jens Schulz, and Andreas Wiese. A constant factor approximation algorithm for unsplittable flow on paths. In *FOCS*, pages 47–56, 2011.
- 11 Adam L. Buchsbaum, Howard J. Karloff, Claire Kenyon, Nick Reingold, and Mikkel Thorup. OPT versus LOAD in dynamic storage allocation. In *STOC*, pages 556–564, 2003.
- 12 Parinya Chalermsook and Bartosz Walczak. Coloring and maximum weight independent set of rectangles. In *SODA*, pages 860–868, 2021.
- 13 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.
- 14 Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms (TALG)*, 3(3):27, 2007.
- 15 Miroslav Chlebík and Janka Chlebíková. Complexity of approximating bounded variants of optimization problems. *Theoretical Computer Science*, 354(3):320–338, 2006.
- 16 Miroslav Chlebík and Janka Chlebíková. Hardness of approximation for orthogonal rectangle packing and covering problems. *Journal of Discrete Algorithms*, 7(3):291–305, 2009.
- 17 Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.
- 18 Julia Chuzhoy and Shi Li. A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2. *Journal of the ACM*, 63(5):1–51, 2016.
- 19 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.
- 20 Max A. Deppert, Klaus Jansen, Arindam Khan, Malin Rau, and Malte Tutas. Peak demand minimization via sliced strip packing. In *APPROX/RANDOM*, volume 207, pages 21:1–21:24, 2021.
- 21 Khaled M. Elbassioni, Naveen Garg, Divya Gupta, Amit Kumar, Vishal Narula, and Arindam Pal. Approximation algorithms for the unsplittable flow problem on paths and trees. In *FSTTCS*, pages 267–275, 2012.
- 22 Leah Epstein, Thomas Erlebach, and Asaf Levin. Online capacitated interval coloring. *SIAM Journal on Discrete Mathematics*, 23(2):822–841, 2009.

- 23 Thomas Erlebach and Klaus Jansen. Off-line and on-line call-scheduling in stars and trees. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 199–213. Springer, 1997.
- 24 Thomas Erlebach and Klaus Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255(1-2):33–50, 2001.
- 25 Waldo Gálvez, Fabrizio Grandoni, Afrouz Jabal Ameli, Klaus Jansen, Arindam Khan, and Malin Rau. A tight $(3/2+\epsilon)$ approximation for skewed strip packing. In *APPROX/RANDOM*, pages 44:1–44:18, 2020.
- 26 Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via l-packings. In *FOCS*, pages 260–271, 2017.
- 27 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, and Arindam Khan. Improved pseudo-polynomial-time approximation for strip packing. In *FSTTCS*, pages 9:1–9:14, 2016.
- 28 Waldo Gálvez, Fabrizio Grandoni, Arindam Khan, Diego Ramírez-Romero, and Andreas Wiese. Improved approximation algorithms for 2-dimensional knapsack: Packing into multiple l-shapes, spirals, and more. In *SoCG*, volume 189, pages 39:1–39:17, 2021.
- 29 Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy, and Andreas Wiese. A $(2 + \epsilon)$ -approximation algorithm for maximum independent set of rectangles. *arXiv preprint*, 2021. [arXiv:2106.00623](https://arxiv.org/abs/2106.00623).
- 30 Jordan Gergov. Algorithms for compile-time memory optimization. In *SODA*, pages 907–908, 1999.
- 31 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. A PTAS for unsplittable flow on a path. In *STOC*, pages 289–302, 2022.
- 32 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. Unsplittable flow on a path: The game! In *SODA*, pages 906–926, 2022.
- 33 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.
- 34 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 *STOC*, pages 607–619, 2018.
- 35 Hamidreza Jahanjou, Erez Kantor, and Rajmohan Rajaraman. Improved algorithms for scheduling unsplittable flows on paths. In *ISAAC*, pages 49:1–49:12, 2017.
- 36 Klaus Jansen, Arindam Khan, Marvin Lira, and K. V. N. Sreenivas. A PTAS for packing hypercubes into a knapsack. In *ICALP*, volume 229, pages 78:1–78:20, 2022.
- 37 Klaus Jansen and Guochuan Zhang. On rectangle packing: maximizing benefits. In *SODA*, pages 204–213, 2004.
- 38 Arindam Khan. *Approximation algorithms for multidimensional bin packing*. PhD thesis, Georgia Institute of Technology, 2015.
- 39 Arindam Khan, Aditya Lonkar, Arnab Maiti, Amatya Sharma, and Andreas Wiese. Tight approximation algorithms for two-dimensional guillotine strip packing. In *ICALP*, volume 229, pages 80:1–80:20, 2022.
- 40 Arindam Khan, Arnab Maiti, Amatya Sharma, and Andreas Wiese. On guillotine separable packings for the two-dimensional geometric knapsack problem. In *SoCG*, volume 189, pages 48:1–48:17, 2021.
- 41 Arindam Khan and Madhusudhan Reddy Pittu. On guillotine separability of squares and rectangles. In *APPROX/RANDOM*, pages 47:1–47:22, 2020.
- 42 Arindam Khan and Eklavya Sharma. Tight approximation algorithms for geometric bin packing with skewed items. In *APPROX/RANDOM*, volume 207, pages 22:1–22:23, 2021.
- 43 Arindam Khan and Mohit Singh. On weighted bipartite edge coloring. In *FSTTCS*, pages 136–150, 2015.
- 44 Joseph SB Mitchell. Approximating maximum independent set for rectangles in the plane. In *FOCS*, pages 339–350, 2022.

- 45 Tobias Mömke and Andreas Wiese. A $(2+\varepsilon)$ -approximation algorithm for the storage allocation problem. In *ICALP*, pages 973–984, 2015.
- 46 Tobias Mömke and Andreas Wiese. Breaking the barrier of 2 for the storage allocation problem. In *ICALP*, pages 86:1–86:19, 2020.
- 47 NS Narayanaswamy. Dynamic storage allocation and on-line colouring interval graphs. In *COCOON*, pages 329–338. Springer, 2004.
- 48 Christos Nomikos, Aris Pagourtzis, and Stathis Zachos. Routing and path multicoloring. *Inf. Process. Lett.*, 80(5):249–256, 2001.
- 49 Arindam Pal. Approximation algorithms for covering and packing problems on paths. *arXiv preprint*, 2014. [arXiv:1402.1107](https://arxiv.org/abs/1402.1107).
- 50 Arka Ray. There is no aptas for 2-dimensional vector bin packing: Revisited. *arXiv preprint*, 2021. [arXiv:2104.13362](https://arxiv.org/abs/2104.13362).
- 51 Peter Winkler and Lisa Zhang. Wavelength assignment and generalized interval graph coloring. In *SODA*, pages 830–831, 2003.
- 52 Gerhard J Woeginger. There is no asymptotic ptas for two-dimensional vector packing. *Information Processing Letters*, 64(6):293–297, 1997.

Optimizing Safe Flow Decompositions in DAGs

Shahbaz Khan ✉ 🏠 

Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

Alexandru I. Tomescu ✉ 🏠 

Department of Computer Science, University of Helsinki, Finland

Abstract

Network flow is one of the most studied combinatorial optimization problems having innumerable applications. Any flow on a directed acyclic graph G having n vertices and m edges can be decomposed into a set of $O(m)$ paths. The applications of such a flow decomposition range from network routing to the assembly of biological sequences. However, in some applications, each solution (decomposition) corresponds to some particular data that generated the original flow. Given the possibility of multiple optimal solutions, no optimization criterion ensures the identification of the correct decomposition. Hence, recently flow decomposition was studied [RECOMB22] in the Safe and Complete framework, particularly for RNA Assembly. The proposed solution reported *all* the *safe* paths, i.e., the paths which are subpath of every possible solution of flow decomposition.

They presented a characterization of the safe paths, resulting in an $O(mn + out_R)$ time algorithm to compute all safe paths, where out_R is the size of the raw output reporting each safe path explicitly. They also showed that out_R can be $\Omega(mn^2)$ in the worst case but $O(m)$ in the best case. Hence, they further presented an algorithm to report a concise representation of the output out_C in $O(mn + out_C)$ time, where out_C can be $\Omega(mn)$ in the worst case but $O(m)$ in the best case.

In this work, we study how different safe paths interact, resulting in optimal output-sensitive algorithms requiring $O(m + out_R)$ and $O(m + out_C)$ time for computing the existing representations of the safe paths. Our algorithm uses a novel data structure called *Path Tries*, which may be of independent interest. Further, we propose a new characterization of the safe paths resulting in the *optimal* representation of safe paths out_O , which can be $\Omega(mn)$ in the worst case but requires optimal $O(1)$ space for every safe path reported. We also present a near-optimal algorithm to compute all the safe paths in $O(m + out_O \log n)$ time. The new representation also establishes tighter worst case bounds $\Theta(mn^2)$ and $\Theta(mn)$ bounds for out_R and out_C (along with out_O), respectively.

Overall we further develop the theory of safe and complete solutions for the flow decomposition problem, giving an optimal algorithm for the explicit representation, and a near-optimal algorithm for the optimal representation of the safe paths.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Network flows; Theory of computation → Network flows; Networks → Network algorithms

Keywords and phrases safety, flows, networks, directed acyclic graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.72

Related Version *Full Version*: <https://arxiv.org/abs/2102.06480>

Funding This work was partially funded by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFE BIO) and partially by the Academy of Finland (grants No. 322595, 328877).

Acknowledgements We would like to thank Manuel Cáceres from University of Helsinki, for helpful discussions and for highlighting several errors in our previous approaches.



© Shahbaz Khan and Alexandru I. Tomescu;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 72; pp. 72:1–72:17

Leibniz International Proceedings in Informatics



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

1 Introduction

Network flow is one of the most studied problems in theoretical computer science with innumerable applications. For a flow network with a unique source s and a unique sink t , every valid flow can be decomposed into a set of weighted s - t paths and cycles [7]. For a directed acyclic graph (DAG) such a decomposition contains only paths. Such path (and cycle) view of a flow indicates how information optimally passes from s to t , being a key step in network routing problems (e.g. [10, 6, 9, 15]), transportation problems (e.g. [16, 17]), or in the more recent and prominent application of reconstructing biological sequences (*RNA transcripts*, see e.g. [18, 23, 8, 5, 22, 27], or *viral quasi-species genomes*, see e.g. [2, 1]).

Finding the minimum flow decomposition (i.e., having the minimum number of paths and cycles) is NP-hard, even if the flow network is a DAG [25]. This hardness result led to research on approximation algorithms [9, 21, 19, 15, 3], and FPT algorithms [12]. Practical approaches usually employ the standard *greedy width* heuristic [25], repeatedly removing an s - t path carrying the most amount of flow. Recently, another pseudo-polynomial-time heuristic was proposed [20] for biological data, which tries to iteratively simplify the graph such that the flow decomposition problem can be solved locally at some vertices.

In the routing and transportation applications, an optimal flow decomposition indicates how to send some information from s to t , and thus any optimal decomposition is satisfactory. However, this is not the case in the prominent application of reconstructing biological sequences, since each flow path represents a reconstructed sequence: a different optimal set of flow paths encodes different biological sequences, which may differ from the real ones. For a concrete example, consider the following application. In complex organisms, a gene may produce more RNA molecules (*RNA transcripts*, i.e., strings over an alphabet of four characters), each having a different abundance. Currently, given a sample, one can read the RNA transcripts and find their abundances using *high-throughput sequencing* [26]. This technology produces short overlapping substrings of the RNA transcripts. The main approach for recovering the RNA transcripts from such data is to build an edge-weighted DAG from these fragments and to transform the weights into flow values by various optimization criteria, and then to decompose the resulting flow into an “optimal” set of weighted paths (i.e., the RNA transcripts and their abundances in the sample) [14]. Clearly, if there are multiple optimal flow decomposition solutions, then the reconstructed RNA transcripts may not match the original ones, and thus be incorrect. Thus, the best possible solution is to find *whatever* can be *safely* reported as being *correct*.

1.1 Problem Definition and Related Work

Recently, Ma et al. [13] were the first to address the issue of multiple solutions to the flow decomposition problem, under a probabilistic framework. Later, they [28] solve a problem (*AND-Quant*), which, in particular, leads to a quadratic-time algorithm for the following problem: given a flow in a DAG, and edges e_1, e_2, \dots, e_k , decide if in *every* flow decomposition there is always a decomposed flow path passing through all of e_1, e_2, \dots, e_k . Thus, by taking the edges e_1, e_2, \dots, e_k to be the edges of a path p , the AND-Quant problem can decide if a path p (i.e., a given biological sequence) appears in all flow decompositions. This indicates that p is likely part of some original RNA transcript.

Another popular approach to address the issue of multiple solutions is the *safety* framework, which was introduced by Tomescu and Medvedev [24] for the genome assembly problem from bioinformatics. For a problem admitting multiple solutions, a partial solution is said to be *safe* if it appears in all solutions to a problem. For the flow decomposition problem, a

path p is safe if for *any* flow decomposition into paths $\mathcal{P} = \{p_1, \dots, p_k\}$, it holds that p is a subpath of some p_i . Considering the weight, a path p is further called *w-safe* if, in *any* flow decomposition, p is a subpath of some path(s) in \mathcal{P}_f whose total weight is at least w .

Khan et al. [11] built upon the AND-Quant problem by addressing flow decomposition under the *safety* framework. They presented a local characterization of safe flow paths as compared to the global characterization of AND-Quant. It was directly adaptable to give an optimal verification algorithm, and a simple enumeration algorithm enumerating all safe paths in $O(mn + out)$ time by applying the characterization on a candidate flow decomposition using the standard *two pointer algorithm*¹. They presented the maximal safe paths out in two formats, the raw output out_R reported each safe path explicitly, and a concise representation out_C which combined the safe paths occurring contiguously in the candidate flow decomposition. Using a worst case example they also proved that the size of out_R can be $\Omega(mn^2)$ in the worst case and $O(m)$ in the best case, whereas that of out_C can be $\Omega(mn)$ in the worst case and $O(m)$ in the best case. However, in their solution the concise representation of the solution depends on the underlying candidate solution used, which hence does not optimize the concise representation. Moreover, they did not address whether the concise representation is the most succinct approach to represent the safe paths.

1.2 Our results

Our main contributions can be described as follows:

1. **Merge-Diverge Property of safe paths.** We develop the theory of safe paths for flow decomposition further by studying the conditions for interaction of safe paths. We prove that two safe paths cannot *merge* at a vertex (or a set of vertices) and later *diverge*.
2. **Optimal output-sensitive enumeration algorithms for the current representations.** We use the *merge-diverge* property to present optimal output-sensitive algorithms for enumerating all safe paths explicitly in $O(m + out_R)$ time and their optimal concise representation in $O(m + out_C)$ time. Our algorithms uses a novel application of the Trie on paths, referred as *Path Tries* which may be of independent interest.
3. **Optimal representation of safe paths.** We present a novel characterization of safe paths out_O allowing us to represent a safe path optimally, requiring $O(1)$ space for every reported path.
 - **Remark 1.** In the worst case both concise representation out_C [11] and our optimal representation out_O may require $\Omega(mn)$ space, however space required per reported path can be much larger for out_C than the optimal $O(1)$ of out_O .
4. **Near optimal algorithm for the optimal output format.** We present an algorithm to report all safe paths using the optimal representation in $O(m + out_O \log n)$ time.
5. **Tighter worst case bounds on out_R and out_C .** Our characterization allows us to prove matching upper bounds for the worst case lower bounds [11] on out_R and out_C .

2 Preliminary

Consider a directed acyclic flow graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, where each edge e has a flow (or weight) $f(e)$ passing through it. For simplicity we assume the graph is connected giving $m \geq n - 1$. For each vertex u , $f_{in}(u)$ and $f_{out}(u)$ denotes the

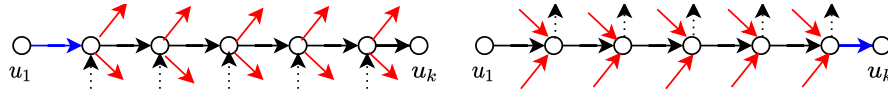
¹ Along a sample solution, the keeping the left end at the start, the right end is moved along the solution as long as the path is safe (evaluated using verification algorithm). This is reported as a maximal safe. Then right end is extended by an edge making the path unsafe, followed by moving the left pointer right until it is safe again. The process is repeated to report the next maximal safe path, and so on.

total flow on its incoming edges and total flow on its outgoing edges, respectively. A vertex v in the graph is called a *source* if $f_{in}(v) = 0$ and a *sink* if $f_{out}(v) = 0$. The set of sources and sinks of the graph G is denoted by $Source(G)$ and $Sink(G)$ respectively. Every other vertex v satisfies the *conservation of flow* $f_{in}(v) = f_{out}(v)$, making the graph a *flow graph*.

For the vertex u , $f_{max}(\cdot, u)$ (or $f_{max}(u, \cdot)$) denotes the maximum value of flow on the incoming edges (or outgoing edges) of u . The corresponding edge is represented by $e_{max}(\cdot, u)$ (or $e_{max}(u, \cdot)$) and its other endpoint (except u) is represented by $v_{max}(\cdot, u)$ (or $v_{max}(u, \cdot)$). Note that in case multiple incoming edges (or outgoing edges) have the maximum flow value, we prefer the edge whose other endpoint (except u) appears first in the topological order, making $e_{max}(\cdot, u)$ (or $e_{max}(u, \cdot)$) and $v_{max}(\cdot, u)$ (or $v_{max}(u, \cdot)$) distinct. Hence, it is referred as *preferred* maximum incoming (or outgoing) edge/vertex. Further, we represent $e_{max}^*(\cdot, u)$ (or $e_{max}^*(u, \cdot)$) as the *unique* maximum incoming (or outgoing) edge if $f_{max}(\cdot, u)$ (or $f_{max}(u, \cdot)$) corresponds to exactly one edge making it equal to $e_{max}(\cdot, u)$ (or $e_{max}(u, \cdot)$) in such a case, and null otherwise. We similarly define its other endpoint (except u) $v_{max}^*(\cdot, u)$ (or $v_{max}^*(u, \cdot)$) which is called *unique* maximum incoming (or outgoing) vertex.

For a path p in the graph, $|p|$ represents the number of its edges. A vertex u is called as being on the *left* of a vertex v on the path, if v is reachable from u on the path. Similarly, in such a case the vertex v is called as being on the *right* of a vertex u on the path. For any path p (or edge) we define its *left extension* to be a path created from p by repeatedly prepending the path with the unique maximum incoming edge of the first vertex of the (updated) path. Similarly, we define the *right extension* of a path to be a path created by repeatedly adding the unique maximum outgoing edge of the last vertex of the (updated) path.

The *flow decomposition* of G is a set of weighted *paths* \mathcal{P}_f such that the flow on each edge in the G equals the sum of the weights of the paths containing it. A path p is called *w-safe* if, in every possible flow decomposition, p is a subpath of some paths in \mathcal{P}_f whose total weight is at least w . A w -safe path with $w > 0$, is called a *safe flow path*, or simply *safe path*. A safe path is *left maximal* (or *right maximal*) if extending it to the left (or right) with any edge makes it unsafe. A safe path is *maximal* if it is both left and right maximal. The safety of a path can be characterized by its *excess flow* (see Figure 1) and properties of safe paths, described as follows.



■ **Figure 1** The excess flow of a path is the incoming or outgoing flow (*blue*) that passes through the path despite the flow (*red*) leaking at its internal vertices (reproduced from [11]).

► **Definition 2** (Excess flow [11]). *The excess flow f_p of a path $p = \{u_1, u_2, \dots, u_k\}$ is*

$$f_p = f(u_1, u_2) - \sum_{\substack{u_i \in \{u_2, \dots, u_{k-1}\} \\ v \neq u_{i+1}}} f(u_i, v) = f(u_{k-1}, u_k) - \sum_{\substack{u_i \in \{u_2, \dots, u_{k-1}\} \\ v \neq u_{i-1}}} f(v, u_i)$$

where the former and later equations are called *diverging* and *converging criterion*, respectively.

► **Theorem 3** (Safe flow paths [11]). *Safety of flow decomposition satisfy the following.*

- (a) A path p is w -safe iff its excess flow $f_p \geq w > 0$.
- (b) The converging and diverging criteria for a path $p = \{u_1, \dots, u_k\}$ are equivalent to

$$f_p = \sum_{i=1}^{k-1} f(u_i, u_{i+1}) - \sum_{i=2}^{k-1} f_{out}(u_i) = \sum_{i=1}^{k-1} f(u_i, u_{i+1}) - \sum_{i=2}^{k-1} f_{in}(u_i).$$

- (c) Adding an edge (u, v) to the start or the end of a path in the flow graph, reduces its excess flow by $f_{in}(v) - f(u, v)$, or $f_{out}(u) - f(u, v)$, respectively.

Additionally, we use the following data structure for answering the level ancestor queries.

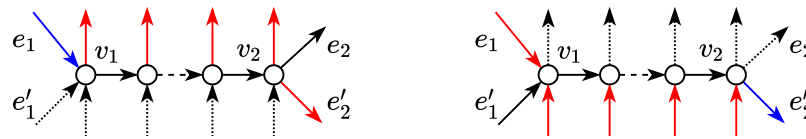
- **Theorem 4** (Level Ancestors [4]). A given tree with n vertices can be preprocessed in $O(n)$ time to report the level ancestor $LA(v, d)$ for a vertex v at a depth d in $O(1)$ time.

3 Interaction of Safe Paths

The previous work [11] focused on properties of safe paths useful for applying the characterization directly in verification and enumeration algorithms. We now explore further properties of safe walks particularly related to the interaction of safe paths and its consequences.

- **Lemma 5** (Merge Diverge). Two safe paths cannot merge (through distinct edges) at an intermediate vertex (or vertices) and then diverge (through distinct edges).

Proof. Let two safe paths p and p' merge at a vertex v_1 , entering v_1 respectively by distinct edges e_1 and e'_1 , and then diverge at a vertex v_2 , leaving v_2 respectively by distinct edges e_2 and e'_2 (see Figure 2).



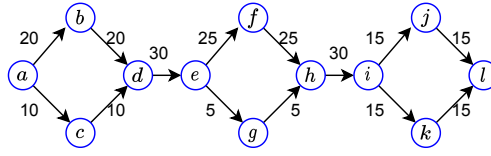
■ **Figure 2** The diverging and converging criterion applied to path p and p' respectively.

Using Theorem 3(c) we know that removing an edge from the end of a path increases the excess flow and hence remains safe. Thus, the subpaths $\{e_1 \cdots, e_2\}$ and $\{e'_1, \cdots, e'_2\}$ of safe paths p and p' respectively, are also safe. By diverging criterion of the safe path p we have $f(e_1) > f(e'_2)$. On the other hand, by converging criterion of the safe path p' we have $f(e'_2) > f(e_1)$, which is a contradiction. ◀

This *merge-diverge* property has an interesting consequence on the structure of a safe path having an edge that is not a unique maximum outgoing edge of a vertex.

- **Lemma 6.** Any safe path having an edge $e(u, v) \neq e_{max}^*(u, \cdot)$, can be extended to the left using only unique maximum incoming edges.

Proof. Consider a path p containing $e_1(u, v) \neq e_{max}^*(u, \cdot)$, which extends to the left of u using edges containing an edge which is not a unique maximum incoming edge $e_2(x, y) \neq e_{max}^*(\cdot, y)$. Now, using Theorem 3(c) we know subpath created by removing edges from the end is safe, as removing such edges only increases the excess flow. Hence, the subpath $p : \{e_2 \cdots e_1\}$ is safe. Further, Theorem 3(c) also implies that a path p' replacing (x, y) with an alternate $e'_2 = e_{max}(\cdot, y)$, and (u, v) with an alternate $e'_1 = e_{max}(u, \cdot)$ is also safe, as we replace an edge with another having at least the same weight. Note that e'_1 and e'_2 always exists since e_1 and e_2 are not unique maximum edges. Thus, both $p : \{e_2 \cdots e_1\}$ and $p' : \{e'_2 \cdots e'_1\}$ are safe which merge at y and then diverge at u using distinct edges, which is a contradiction. ◀



■ **Figure 3** Problem with the simplistic approach. While processing the vertex e , we get two safe paths $p_1 : \langle a, b, d, e \rangle$ and $p_2 : \langle a, c, d, e \rangle$. As we continue processing to reach h , both p_1 and p_2 are extended through f to get $p_{11} : \langle a, b, d, e, f, h, i \rangle$ and $p_{21} : \langle a, c, d, e, f, h, i \rangle$. However, when extending through g both get trimmed to give the same $p_{12}, p_{22} : \langle d, e, g, h, i \rangle$. Further, the paths p_{11} and p_{22} are again extended through j and k (recall the two pointer algorithm) to get four paths, $p_{111}, p_{211} : \langle d, e, f, h, i, j, l \rangle$ and $p_{112}, p_{212} : \langle d, e, f, h, i, k, l \rangle$. Moreover, paths p_{12}, p_{22} can also be extended through j and k to get $p_{121}, p_{221} : \langle h, i, j, l \rangle$ and $p_{122}, p_{222} : \langle h, i, k, l \rangle$. We thus obtain duplicate paths representing the same safe paths from different sources, and some non-maximal paths (p_{122}, p_{222}) which are subpath of the other reported paths.

4 Optimal computation of Raw Safe paths

The essential bottle-neck of the previous approach [11] was the use of a candidate flow decomposition, on whose subpaths the safety criteria was evaluated. The computation of a candidate flow decomposition itself requires $O(mn)$ time making it suboptimal. In order to avoid it we are required to process the graph in a structured manner. Given the graph is a DAG, the topological ordering of the graph serves this purpose.

A simple approach is to follow the topological order and maintain all maximal safe paths ending at the currently processed vertex explicitly. And use the two pointer algorithm to extend it as we continue processing the vertices in the topological order. However, to avoid duplicate and non-maximal results we need to identify the common suffixes of the safe paths, which can be processed accordingly (see Figure 3). Fortunately, for strings the data structure Trie (considered on reversed strings) serves exactly for the same purpose which motivates us to use Tries for storing all the left maximal safe paths ending at a vertex as follows.

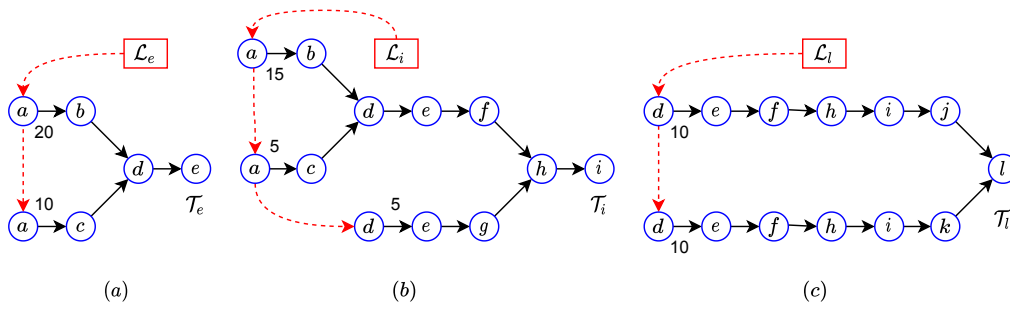
4.1 Data structures \mathcal{T}_u and \mathcal{L}_u

We build a Trie structure \mathcal{T}_u treating the reverse paths ending at a vertex u as strings, such that the common suffixes of the paths are combined. Note that a vertex v can appear multiple times in the Path Trie, if multiple paths containing v do not share v in their common suffix.

All left maximal safe paths ending at a vertex u are hence maintained in \mathcal{T}_u . Additionally, we maintain a linked list \mathcal{L}_u of the leaves of \mathcal{T}_u along with the path's corresponding excess flow, i.e. $\mathcal{L}_u = \{(v_1, f_1), (v_2, f_2), \dots\}$. Consider Figure 4, we show the path tries at the vertices e, i and l for the graph shown in Figure 3. Note that the leaves represent the left maximal safe paths without repetition or storing subpaths as in the simplistic approach.

4.2 Algorithm

The main idea behind our approach is to uniquely extend each safe path ending at a vertex to its preferred maximum out-neighbour in constant time associated with each edge. For the rest of the out-neighbours, we can build their safe paths from scratch at the expense of the path length. This requires us to process the vertices in the topological order of the graph, such that all the safe paths ending at a vertex are computed before it is processed. We maintain the left maximal safe paths ending at a vertex u in the Path Trie \mathcal{T}_u and the list of safe paths \mathcal{L}_u . Our algorithm uses optimal $O(m + out_R)$ time, where out_R is the size of the raw output, i.e., each safe path stored explicitly, which is optimal.



■ **Figure 4** Path Trie structure storing the left maximal paths ending at vertices (a) \mathcal{T}_e with \mathcal{L}_e , (b) \mathcal{T}_i and \mathcal{L}_i , and (c) \mathcal{T}_l and \mathcal{L}_l .

Algorithm 1 describes our approach, where vertices are processed in topological order such that while processing a vertex u , all the left maximal safe paths ending at u (along with their excess flow) are stored in \mathcal{T}_u and \mathcal{L}_u . While processing u all its safe paths are evaluated for a possible extension to its preferred maximum outgoing neighbour $v^* = v_{max}(u, \cdot)$, which always exists except when u is a sink. Note that v^* is not the unique maximum outgoing neighbour ($v_{max}^*(u, \cdot)$) rather preferred ($v_{max}(u, \cdot)$), so that \mathcal{T}_u can be used to extend to some vertex in case there is no unique maximum. Additionally, when u is a source, we have an empty \mathcal{T}_u , so we have no safe paths to extend. Hence, for the other cases we check all paths in \mathcal{L}_u for possible extension to v^* using Theorem 3(c). For this we add the complete \mathcal{T}_u as a child of \mathcal{T}_{v^*} . Thereafter, we need to trim the prefixes of paths in \mathcal{T}_{v^*} which are not safe and compute the list of safe paths \mathcal{L}_{v^*} . Further, we need to add the safe paths using every other outgoing edge (u, v) ($v \neq v^*$) in the corresponding \mathcal{T}_v .

We now process each path in \mathcal{L}_u . The paths which are not safe on extending to (u, v^*) are clearly right-maximal (and hence maximal), and hence are reported in the solution SOL. We extend all the paths in \mathcal{L}_u with (u, v^*) , and add their maximal suffixes which are safe to \mathcal{T}_{v^*} . Note that the entire path may be safe or at least the edge (u, v^*) is safe. So we start trimming \mathcal{T}_{v^*} until the path is safe. This is done by maintaining in f_x the excess flow of the path from x to v^* , where f_x is updated using Theorem 3(c). When (u, v^*) is added f_x may become negative in which case the path is trimmed from the left until f_x is positive. Now, since in a Trie an edge can be shared by multiple paths having a common suffix, we trim the edge only if it is a leaf, and similarly add to \mathcal{L}_{v^*} only if it starts from a leaf. Hence, multiple paths from \mathcal{L}_v will not add the same safe path to \mathcal{L}_{v^*} as it will start from a leaf only when the last such path is processed. This also avoids adding a non-maximal path which is a subpath of another safe path.

Finally, we need to add the safe paths to non-preferred maximum outgoing neighbours, which by Lemma 6 is always on a single path containing the unique maximum incoming edges. We thus compute the single safe left extension for all such neighbours explicitly using Theorem 3(c). We do this again by maintaining the excess flow of the path from x to v in f_x . As we start the path is a single edge with f_x necessarily positive, where we continue adding the preferred maximum incoming edge to the left until f_x is negative. Note that we do not insist on a unique maximum incoming edge as required by Lemma 6 as the flow f_x will itself become negative if the preferred maximum incoming edge is not unique.

■ **Algorithm 1** Optimally Computing Raw representation of Safe Paths.

```

Compute Topological Order of  $G$ 
forall  $u \in V$  in topological order do
  COMPUTE-SAFE( $u$ )
COMPUTE-SAFE( $u$ ):
if  $u \notin \text{Sink}(G) \cup \text{Source}(G)$  then
  |  $v^* \leftarrow v_{\max}(u, \cdot)$ 
else  $v^* \leftarrow \text{null}$ 
if  $u \in \text{Source}(G)$  then Initialize  $\mathcal{T}_u$  with  $u$ 
forall  $(u, v) \in G, v \neq v^*$  do // new paths
  | Add  $(u, v)$  to  $\mathcal{T}_v$ 
  |  $x \leftarrow u, f_x \leftarrow f(u, v)$ 
  | while  $x \notin \text{Source}(G)$  and  $f_x - f_{\text{in}}(x) +$ 
  |    $f_{\max}(\cdot, x) > 0$  do
  |   | Add  $e_{\max}(\cdot, x)$  to  $\mathcal{T}_v$ 
  |   |  $f_x \leftarrow f_x - f_{\text{in}}(x) + f_{\max}(\cdot, x)$ 
  |   |  $x \leftarrow v_{\max}(\cdot, x)$ 
  | Add  $(x, f_x)$  to  $\mathcal{L}_v$ 
if  $v^* \neq \text{null}$  then
  | Make  $\mathcal{T}_u$  as child of  $v^*$  in  $\mathcal{T}_{v^*}$ 
forall  $(x, f_x) \in \mathcal{L}_u$  do // Process  $\mathcal{T}_u$ 
if  $v^* = \text{null}$  or  $f_x - f_{\text{out}}(u) + f(u, v^*) \leq 0$ 
then
  |  $p \leftarrow$  Extract path from  $x$  to  $u$  in  $\mathcal{T}_u$ 
  | Add  $(p, f_x)$  to SOL
if  $v^* \neq \text{null}$  then
  |  $f_x \leftarrow f_x - f_{\text{out}}(u) + f(u, v^*)$ 
  | while  $f_x \leq 0$  and  $x$  is leaf of  $\mathcal{T}_{v^*}$  do
  |   |  $y \leftarrow$  Parent of  $x$  in  $\mathcal{T}_{v^*}$ 
  |   |  $f_x \leftarrow f_x + f_{\text{in}}(y) - f(x, y)$ 
  |   | Remove  $(x, y)$  from  $\mathcal{T}_{v^*}$ 
  |   |  $x \leftarrow y$ 
  | if  $x$  is a leaf in  $\mathcal{T}_{v^*}$  then
  |   | Add  $(x, f_x)$  to  $\mathcal{L}_{v^*}$ 

```

4.3 Correctness

We prove the correctness of the algorithm by induction over the topological order of the graph. The underlying invariant is as follows:

After COMPUTE-SAFE(u) is executed, all left maximal safe paths having starting vertex and the internal vertices with topological order up to u , are stored in corresponding \mathcal{T}_v and \mathcal{L}_v . Also, all maximal safe paths ending at vertices with topological order up to u , are reported in SOL.

The base case is trivially true when no vertices are processed, as no safe paths exist. Now, when we start processing u , using the invariant we know all the *left maximal* safe paths not having u as internal vertex, and all the *maximal* safe paths ending at the vertex with topological order less than u are already in SOL. So we need to process only the *left maximal* paths having u , which necessarily have the last internal vertex as u , and all the *maximal* safe paths ending at u must be added to SOL. The prefix (not necessarily proper) of both these kinds of paths up to u , are clearly safe (using Theorem 3(c)) and hence are present in \mathcal{T}_u and \mathcal{L}_u by the invariant.

Now, all the *left maximal* safe paths are checked for a possible extension to v^* by construction, and for the remaining out-neighbours we explicitly add the single safe path possible (Lemma 6). Further, all the paths in \mathcal{L}_u are checked for being maximal and added to SOL in such a case. Note that processing the vertices in the topological order ensures that all safe paths ending at u have internal vertices already processed so that the complete path is present in \mathcal{T}_u and \mathcal{L}_u before it is processed.

4.4 Analysis

The total time required by the algorithm can be associated with the edges of the graph m or the total length of safe paths reported, i.e. size of the raw representation of the output out_R . Computing the topological order of the graph requires $O(m)$ time. Now, for each vertex u ,

processing the paths in \mathcal{L}_u either extends it to v^* in $O(1)$ time or reporting a safe path p and extending its subpath to v^* in $O(|p|)$ time. In the former case, the length of the safe path is increased by one (adding u), and the latter case is associated with the length of the reported path (as each safe path is reported exactly once). For residual out-neighbours, the time required is proportional to the size of added safe path. Hence we have the following:

► **Theorem 7.** *Given a flow graph (DAG) having n vertices and m edges, the set of all safe paths can be optimally reported in its raw representation in $O(m + out_R)$ time.*

5 Optimal computation of the optimal Concise Representation

Previous work [11] presented a simple algorithm for computing the concise representation of the solution. Hence instead of reporting each safe path individually which may have overlaps among each other, they combined several overlapping safe paths to report a single path p along with indices representing the subpaths of p which are maximally safe.

However, their concise representation was dependent on the underlying candidate path decomposition, which may be suboptimal. We shall now present an optimal algorithm for computing the optimal concise representation of the solution. Our algorithm again uses Path Tries \mathcal{T}_u with a modified version \mathcal{L}_u^+ of the list of safe paths to store the concise representation.

5.1 Data structures \mathcal{T}_u and \mathcal{L}_u^+

We again build a Trie structure on the reverse paths, whose common suffixes are combined. Similar to the previous algorithm, it stores all the left maximal safe paths ending at u .

Now, the concise representation of safe paths are reported in the form $\{(p_1, I_1), (p_2, I_2), \dots\}$, where each path p_i has maximal safe subpaths denoted by intervals $I_i = \{(l_1, r_1, f_1), (l_2, r_2, f_2), \dots\}$. Each (l_j, r_j) and f_j denote the corresponding end vertices of the maximal safe path on p_i and its excess flow, respectively. While processing u , the partial results are maintained in the list $\mathcal{L}_u^+ = \{(p_1, I_1), (p_2, I_2), \dots\}$ where p_i are the partially built concise representation of the safe paths, where the reported paths contain u . Further, for each I_i the last interval (l_j, r_j, f_j) has $r_j = u$ representing the left maximal safe path ending at u , whereas the remaining intervals are maximal. Note that while processing u , p_i does not include the last path from l_j to u .

5.2 Algorithm

The main idea behind our approach is to always attempt to extend a path p_i of the concise representation with the interval corresponding to a safe path that overlaps the most with p_i . Clearly, while extending the left maximal safe paths on u to its out-neighbours, the maximum such overlap with p_i would correspond to the safe path ending at the preferred maximum outgoing neighbour v^* , which is hence added to p_i . However, in case multiple paths p_i, p_j in the concise representation add exactly the same safe path p_{v^*} corresponding to v^* , it is not optimal to add p_{v^*} to both p_i and p_j . In such a case p_{v^*} can be added to anyone such path (say p_i), and p_j will add the maximum overlapping path p_v corresponding to some other out-neighbour v of u , if it exists. If no such neighbour exists, we will report p_j in the solution having the last interval ending at u . And in case the safe path $p_{v'}$ of some out-neighbour v' of u is not accommodated in the existing paths of the concise representation, we add a new path $p_{v'}$ to the concise representation. The optimality of our concise representation is guaranteed by our choice of maximum overlap, ensuring a new path in the concise representation is always of the minimum length.

Consider Algorithm 2, similar to the previous algorithm we process each vertex in the topological order, and when a vertex u is processed its \mathcal{T}_u and \mathcal{L}_u^+ have already been computed by its incoming neighbours. Similar to the previous approach for each out-neighbour v of u , that is not the preferred maximum out-neighbour of u exactly a single safe path exists containing the unique maximum incoming edges (Lemma 6). We compute it similar to the previous algorithm for each v from scratch, and update its corresponding \mathcal{T}_v , inserting the path p_v temporarily to \mathcal{L}_v^+ as a new path. This path can potentially be added to some existing path in \mathcal{L}_u^+ , for which we mark the starting vertex of p_v in \mathcal{T}_u .

Now, for the unique maximum outgoing neighbour v^* of u , we add \mathcal{T}_u as a child of v^* in \mathcal{T}_{v^*} and attempt to extend each path p_k in \mathcal{L}_u^+ to v^* using Theorem 3(c). Similar to the previous algorithm, this is accompanied by trimming the leaves of last interval of p_k from \mathcal{T}_u if they are not safe in \mathcal{T}_{v^*} . Again, if the start of the safe path x for v^* is no longer a leaf, then the same safe subpath is shared by some other safe path in \mathcal{L}_u^+ . In such a case we do not extend p_k to v^* , rather either (a) extend it to the out-neighbour v of u having the maximum overlap (lowest vertex marked in \mathcal{T}_u along the last interval of p_k) which is not a unique maximum out-neighbour of u , or (b) terminate p_k at u including it as its last interval. Thus, while processing \mathcal{T}_u for each path in \mathcal{L}_u^+ we deal with five distinct cases (see Algorithm 2).

- (a) **v^* is null because $u \in \text{Source}(G)$:** The list \mathcal{L}_u^+ is empty and each out-neighbour v of u is addressed as non-preferred maximum out-neighbour adding the corresponding edge to their \mathcal{T}_v and \mathcal{L}_v^+ computing from scratch.
- (b) **v^* is null because $u \in \text{Sink}(G)$:** For all paths in \mathcal{L}_u^+ , the left limit x reaches u (as f_x is always negative and no out-neighbour exists to mark a vertex in \mathcal{T}_u), which is hence updated to include the last interval of I_k and added to SOL.
- (c) **Path p_k is extended to include v^* :** The safe path is unique to p_k and hence the left limit of safe path x is always a leaf, terminating as soon as $f_x > 0$ and added to $\mathcal{L}_{v^*}^+$ accordingly. Note that no vertex before reaching $f_x > 0$ could have been marked, as left limit of v^* would be the lowest among all out-neighbours of u .
- (d) **Path p_k is extended to include some $v \neq v^*$:** This is possible only if the safe path for v^* is not unique to p_k , i.e., x is no longer a leaf. Then maximum overlap is the lowest vertex along x to u path, to which p_k is added accordingly.
- (e) **Path p_k is not extended and reported in Sol:** This is possible again when safe path for v^* is not unique, so x is no longer a leaf and x reaches u similar to case (b).

5.3 Correctness and Analysis

The optimality of out_C is ensured by appending a path in out_C with the safe path having the maximum overlap with the existing path. This is ensured by processing the marked vertices bottom-up, which represent the start vertex of the safe paths corresponding to the out-neighbour v of u , which is not a unique maximum out-neighbour of u . This guarantees that in case the path cannot be uniquely extended to v^* , the vertex v with the maximum overlap is selected resulting in an optimal concise representation.

The total time taken while processing u is dominated by the processing of \mathcal{L}_u^+ and building the safe paths for those out-neighbours of u which are not unique maximum out-neighbours of u , from scratch. Consider the cases (a), (b), (c) and (e), the time taken in processing \mathcal{L}_u^+ can be easily associated with the length of the path p_k in \mathcal{L}_u^+ since it will be reported exactly once (cases (b) and (e)), and removed from the last interval exactly once (case (c)). Case (a) is also easy to associate as it increases the corresponding paths in \mathcal{L}_v^+ , and hence can be associated with its length.

The only hard case is (d) as it computes the safe path for v from scratch, but extends it on an existing $p_k \in \mathcal{L}_u^+$ which takes more time than the increase in p_k . However, note that this is possible only when the left limit x is no longer a leaf, which implies that the extra processed path is a common suffix of multiple paths in \mathcal{L}_u^+ . And hence the suffix was accounted for only once for multiple paths, and now when p_k is detached from the common suffix the cost of the processed path can be associated with that of the detached path (previously unaccounted being a part of the common suffix). Thus, all the steps can be accounted for with the length of out_C and we get the following.

► **Theorem 8.** *Given a flow graph (DAG) having n vertices and m edges, the optimal concise representation out_C of the safe paths can be optimally reported in $O(m + out_C)$ time.*

■ **Algorithm 2** Optimally Computing concise Representation of Safe Paths.

```

Compute Topological Order of  $G$ 
forall  $u \in V$  in topological order do
  | COMPUTE-SAFE-COMPR( $u$ )

COMPUTE-SAFE-COMPR( $u$ ): if  $u \notin Sink(G) \cup Source(G)$  then
  |  $v^* \leftarrow v_{max}(u, \cdot)$ 
else  $v^* \leftarrow null$ 
if  $u \in Source(G)$  then Initialize  $\mathcal{T}_u$  with  $u$ 
forall  $(u, v) \in G, v \neq v^*$  do // new paths
  | Add  $(u, v)$  to  $\mathcal{T}_v$ 
  |  $x \leftarrow u$  in  $\mathcal{T}_u, f_x \leftarrow f(u, v)$ 
  | while  $x \neq leaf$  of  $\mathcal{T}_u$  and  $f_x - f_{in}(x) + f_{max}(\cdot, x) > 0$  do
  |   | Add  $e_{max}(\cdot, x)$  to  $\mathcal{T}_v$ 
  |   |  $f_x \leftarrow f_x - f_{in}(x) + f_{max}(\cdot, x)$ 
  |   |  $x \leftarrow v_{max}(\cdot, x)$  in  $\mathcal{T}_u$ 
  |   | Add  $(\emptyset, \{(x, v, f_x)\})$  to  $\mathcal{L}_v^+$ 
  |   | Push  $v$  to  $Mark[x]$ 
  |   | Add  $x$  to  $\mathcal{M}$ 
if  $v^* \neq null$  then
  | Add  $\mathcal{T}_u$  as child of  $v^*$  in  $\mathcal{T}_{v^*}$ 

forall  $(p_k, I_k) \in \mathcal{L}_u^+$  do // Process  $\mathcal{T}_u$ 
  |  $(l_i, u, f_i) \leftarrow$  Last of  $I_k, x \leftarrow l_i$  in  $\mathcal{T}_u$ 
  | if  $v^* = null$  then  $f_x \leftarrow -\infty$ 
  | else  $f_x \leftarrow f_i - f_{out}(u) + f(u, v^*)$ 
  | while  $f_x \leq 0$  and  $Mark[x] = \emptyset$  and  $x \neq u$  do
  |   |  $y \leftarrow$  Parent of  $x$  in  $\mathcal{T}_u$ 
  |   | if  $y$  is not a leaf in  $\mathcal{T}_u$  then
  |   |   |  $f_x \leftarrow f_x + f_{in}(y) - f(x, y)$ 
  |   |   | Remove  $(x, y)$  from  $\mathcal{T}_u$ 
  |   |   |  $x \leftarrow y$ 
  |   |  $p \leftarrow$  Path from  $l_i$  to  $x$  in  $\mathcal{T}_u$ 
  |   |  $p_k \leftarrow p_k \cup \{p \setminus \{x\}\}$ 
  |   | if  $f_x > 0$  then
  |   |   | if  $l_i \neq x$  then Add  $(x, v^*, f_x)$  to  $I_k$ 
  |   |   | else Last of  $I_k \leftarrow (l_i, v^*, f_x)$ 
  |   |   | Add  $(p_k, I_k)$  to  $\mathcal{L}_{v^*}^+$ 
  |   | else
  |   |   | if  $Mark[x] \neq \emptyset$  then
  |   |   |   |  $v \leftarrow$  Pop from  $Mark[x]$ 
  |   |   |   |  $(\emptyset, I_v) \leftarrow$  Pop from  $\mathcal{L}_v^+$ 
  |   |   |   | Add  $(p_k, I_v \cup I_k)$  to  $\mathcal{L}_v^+$ 
  |   |   | else Add  $(p_k \cup \{x\}, I_k)$  to SOL
  |   | forall  $x \in \mathcal{M}$  do Clear  $Mark[x]$ 
  |   | Clear  $\mathcal{M}$ 

```

6 Optimal Representation of Safe paths

The raw representation of the safe paths out_R can take $\Theta(mn^2)$ space and hence time in the worst case. The previous work [11] presented a concise representation of the safe paths reporting a combination of the overlapping safe paths along with its indices, requiring total $\Theta(mn)$ space. However, it may not be optimal as the total size of this concise representation may be much larger than the number of safe paths. We thus present an optimal representation of the safe path whose size requires $\Theta(1)$ space for every safe path reported.

6.1 Representative edge with left and right extensions

Lemma 6 presents an interesting property about safe paths being extendible in a preferred way to the left for edges which are not unique maximum outgoing edges of some vertex. We extend the notion further by considering a representative edge for each safe path such that the maximal path can always be generated by extending it to the left along unique maximum incoming edges and to the right along unique maximum outgoing edges as follows.

► **Theorem 9** (Representative edge). *Given a flow graph (DAG), every safe path p can be described using a representative edge e_p , such that p can be constructed by extending e_p to the left along the unique maximum incoming edges and to the right along the unique maximum outgoing edges.*

Proof. Given a maximal safe path p , let $e(x, y) \in p$ be the leftmost edge such that $e(x, y) \neq e_{max}^*(\cdot, y)$. If no such edge exists, we define the last edge as the representative edge of p , where rest of p is along its unique maximum incoming edges (left extension of e) proving the existence of the representative edge.

Now, by definition the prefix of p on the left of e is along the unique maximum incoming edges (left extension of e), so we only need to prove that the suffix of p after e is along the unique maximum outgoing edges (or the right extension of e). We shall prove it by contradiction, hence assume there exist an edge $e'(a, b) \in p$ to the right of e such that $e'(a, b) \neq e_{max}^*(a, \cdot)$. Clearly, the path $e_{max}^*(\cdot, y) \cup p[y, a] \cup e_{max}^*(a, \cdot)$ is also safe using Theorem 3(c). However, this violates the merge-diverge property (Lemma 5) contradicting our assumption and proving the existence of e as the representative edge of p . ◀

► **Remark 10.** Every safe path contains a representative edge which is either the last edge which is also unique maximum incoming edge, or an edge which is not a unique maximum incoming edge. For the sake of uniformity, in case multiple edges satisfy this property (not being unique maximum incoming edge) for a safe path p , we consider e_p to be the rightmost such edge.

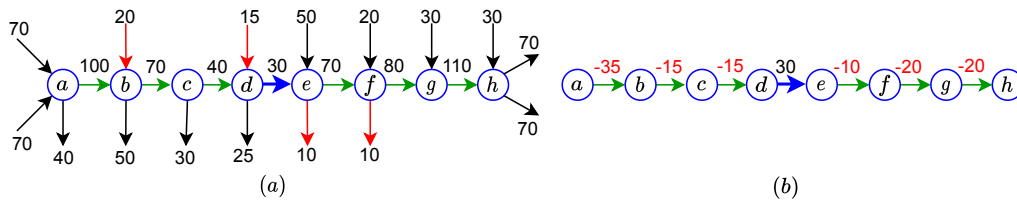
Note that the safe paths when represented using such a representation requires $O(1)$ space per safe path to store the representative edge and its two endpoints, where a single representative edge may store multiple pairs of endpoints representing individual safe paths. We assume that the unique maximum incoming and outgoing edges of each vertex are known which can be pre-computed in $O(m)$ time and stored using $O(n)$ space.

6.2 Approach

As described previously in Remark 10, the representative edge e can be either (a) an edge which is not a unique maximum incoming edge, or (b) a unique maximum incoming edge. The former case is non-trivial as the edge can represent multiple safe paths, whereas the latter is trivial as the edge can represent at most one (can be zero) maximal safe path ending at e . So here we describe only the non-trivial case as trivial can be computed similarly.

Consider Figure 5 (a), where the edge (d, e) is a edge which is not a unique maximum incoming edge of e . The path $\langle a, b, c, d \rangle$ is its left extension (along unique maximum incoming edges), and the path $\langle e, f, g, h \rangle$ is its right extension (along unique maximum outgoing edges). Now, once we compute the left and right extensions we can easily compute all the safe paths represented by (d, e) using two pointer approach in time proportional to length of the path. We get the maximal safe paths $\langle b, c, d, e, f \rangle$ and $\langle d, e, f, g, h \rangle$.

However, assuming we have pre-computed the left and right extensions we can compute all the safe paths using binary search along the path in time $O(\log n)$ times the number of safe paths. This can be done by pre-computing the loss for extension along each edge



■ **Figure 5** Graphs describing (a) the left and right extensions of a representative edge, and (b) the cumulative losses on extension along each edge.

(Theorem 3(c)) and storing the cumulative value on the edge. Now, we find the safe paths as follows. Given the flow on (d, e) is 30 we search for the leftmost minimum value greater than -30 which we find as (b, c) with value -15 giving a left maximal safe path. We make it right maximal (and hence maximal) by searching for the rightmost minimum value greater than $-30 + 15 = -15$ which we find as (e, f) , giving our maximal safe path from b to f using two binary searches. Now, to find the next maximal we extend it to right including (f, g) and again find the left maximal on $-30 + 20$ as (d, e) , and thereafter right maximal on $-30 + 0$ as (g, h) , giving the second maximal safe path from d to h using another two binary searches.

6.3 Data structures

As described in our approach above we require pre-computed left and right extensions for each edge along with the cumulative losses on each edge for efficient search of the maximal safe paths. This can be efficiently computed by building all possible left and right extensions separately which will form two forests corresponding to all unique maximum incoming and outgoing edges. Further for $O(1)$ time access to elements on these forests for binary search we require the classical Level ancestor data structure.

1. Unique maximum incoming and outgoing forests \mathcal{F}_i and \mathcal{F}_o .

For each vertex in the graph, we add the unique maximum incoming edge (if exists) to \mathcal{F}_i . Clearly, each vertex has at most one incoming neighbour (parent) making \mathcal{F}_i a forest. Similarly, for each vertex, we add the *reverse* of the unique maximum outgoing edge (if exists) and \mathcal{F}_o . Again, each vertex has at most one incoming neighbour as a parent (as we added reverse edges), making \mathcal{F}_o .

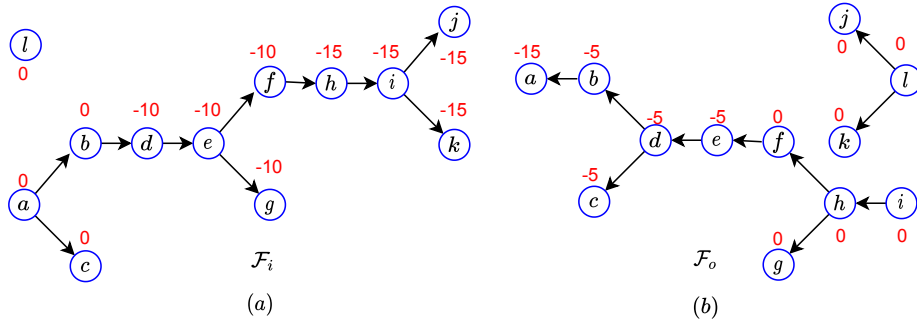
Now, for each vertex v in the forest \mathcal{F}_i we store the cumulative loss $c_i[v]$ (using Theorem 3(c)) along the path v to the root of its tree $root[\mathcal{F}_i(v)]$. The cumulative loss for any subpath from v to u (where u is ancestor of v in \mathcal{F}_i), can be simply computed as $c_i[u] - c_i[v]$. Similarly, we store the cumulative loss for each vertex v on \mathcal{F}_o in $c_o[v]$. The corresponding forests \mathcal{F}_i and \mathcal{F}_o for Figure 3 are shown in Figure 6. Clearly, these structures can be computed in $O(m)$ time.

2. Level ancestors LA_i and LA_o

We use Theorem 4 to compute the data structure on \mathcal{F}_i and \mathcal{F}_o using $O(n)$ time, for reporting the level ancestors $LA_i(v, d)$ and $LA_o(v, d)$ of a vertex v at depth d in $O(1)$ time.

6.4 Algorithm

We now describe how our approach (described on paths) can be used to compute the optimal representation of all the maximal safe paths in $O(m + out_O \log n)$ time.



■ **Figure 6** For the graph in Figure 3 we show (a) the unique maximum incoming forest \mathcal{F}_i and (b) the unique maximum outgoing forest \mathcal{F}_o . Note that l (in \mathcal{F}_i) and i (in \mathcal{F}_o) do not have a parent in the absence of corresponding unique maximum edges.

We first address computing all non-trivial safe paths. Consider each edge which is not unique maximum incoming edge, say $e(u, v) \neq e_{max}^*(\cdot, u)$. We use the two pointer algorithm as described in [11] however in each step we perform a binary search to compute each safe path in $O(\log n)$ time. The binary search computes the excess flow on a path using the values of $c_o[v]$, $c_i[v]$, and probes an element by considering ancestors of u in \mathcal{F}_i and ancestors of v in \mathcal{F}_o which can be directly accessed in $O(1)$ time using LA_i and LA_o structures. The optimal output reports a list of pairs of end vertices for maximal safe paths represented by $e(u, v)$. However, we avoid this algorithm if the maximal safe path containing e is the single edge e , which is typically not reported. This can be evaluated in $O(1)$ time using Theorem 3(c) in both left and right directions.

For computing the trivial paths, we need to compute all maximal safe paths containing only unique maximum incoming edges. We simply look at the leaves of \mathcal{F}_i and search for the left maximal path ending on it, and thereafter continue the search using a modified two pointer algorithm (using binary search) on its ancestors. However, we need to ensure two aspects. Firstly, we do not perform a binary search in case there exist no maximal safe path, which is only possible if (a) $f(u, v) > |c_i[u]|$, implying the entire path to root is safe, and (b) $f(e_{max}(v, \cdot)) > |c_i[v]|$ implying the path till v is not maximal. In case only (a) is true we report a single safe path from the root to v , and if both are true we skip the leaf. Secondly, the two pointer algorithm on internal vertices may repeat the safe paths reported by other leaves as the internal vertices may be shared. Hence we mark the internal vertices which have been processed so as to avoid repetition of processing and results.

6.5 Implementation details

For the sake of completeness we now describe the two pointer algorithm using binary search on \mathcal{F}_i and \mathcal{F}_o in detail.

The algorithm computes all pairs of end points for safe paths represented by $e(u, v)$ as follows. It first computes the start of left maximal safe path ending at v by performing a binary search on the ancestors of u in \mathcal{F}_i . It computes the highest ancestor l of u such that excess flow of l to v is positive, i.e. $c_i[l] < c_i[u] + f(e)$. This search involves accessing the ancestor at mid-depth directly in $O(1)$ time using $LA_i(u, d[u]/2)$ and so on for the whole binary search. Thereafter, the algorithm computes the end r of right maximal path starting from l in ancestors of v in \mathcal{F}_o such that excess flow of the path $\langle l, \dots, u, v, \dots, r \rangle$ is positive, i.e. highest ancestor of v such that $c_o[r] < f(e) + c_i[u] + c_o[v] - c_i[l]$. We record

$[r, l]$ with the corresponding flow $f(e) + c_i[u] - c_i[l] + c_o[v] - c_o[r]$ as a safe path for e . Then we find the next start of the left maximal path ending at ancestor of r in \mathcal{F}_o , i.e., using $f(e) + c_o[v] - c_o[r]$ instead of $f(e)$ in the process described above. This continues until the left maximal reaches v or the right maximal reaches the root of \mathcal{F}_o containing v .

6.6 Correctness and Analysis

The correctness of our algorithm follows from that of the two pointer algorithm described in [11]. By construction, we show that the algorithm requires $O(1)$ time to check whether a safe path exists corresponding to a representative edge, and thereafter $O(\log n)$ time to report each safe path represented by the edge. Since any explicit representation of the safe paths would require $O(1)$ space for every safe path, out_O is the number of safe paths. We thus get the following result.

► **Theorem 11.** *Given a flow graph (DAG) having n vertices and m edges, the optimal representation out_O of the safe paths can be reported in $O(m + out_O \log n)$ time.*

7 Space bounds for different representations of safe paths

Previous work [11] presented a worst case example demonstrating that out_R and out_C can respectively be $\Omega(mn^2)$ and $\Omega(mn)$ in the worst case and $O(m)$ in the best case. The worst case example graph they presented also gives a bound of $\Omega(mn)$ in the worst case and $O(m)$ in the best case for out_O . In the light of the new characterization, we shall now understand these bounds in more detail.

Using Theorem 9, we know that every safe path can be represented as an edge with a subpath of its left and right extensions. Now, in the worst case each of the m edges can have left and right extensions of length $O(n)$ each, making a complete path of $O(n)$ size. Using the two pointer algorithm we know, that the number of maximal safe paths on a path p are $|p|$. Hence, for each edge we can have $O(n)$ safe paths, each of possibly $O(n)$ edges in the worst case.

This establishes an upper bound of $O(mn^2)$ on the size of out_R matching the $\Omega(mn^2)$ bound of [11], proving the tight bound of $\Theta(mn^2)$ on out_R in the worst case. Further, since each edge with its extensions creates a valid path for out_C , having $O(n)$ length and $O(n)$ indices on every path, we also get $O(mn)$ bound for out_C and out_O , resulting in tight worst case bound of $\Theta(mn)$ for both out_C and out_O .

8 Conclusion

We study the optimization of the solutions for the safety of flow paths in a given flow graph (DAG), which has applications in various domains, including the more prominent assembly of biological sequences. The previous work characterized such paths giving an optimal verification algorithm but suboptimal enumeration algorithms, which required computing a candidate flow decomposition taking $\Omega(mn)$ time even when the reported solution is small.

We present output-sensitive optimal algorithms for reporting the safe paths when represented in the raw format reporting each path explicitly, and optimal concise representation previously described. This is achieved by exploiting a crucial property related to the interaction of safe paths and a novel data structure Path Tries, which may be of independent interest. Further, we characterized an optimal representation of the safe paths, requiring $O(1)$ space for every safe path reported. We also presented a near optimal algorithm to

compute the optimal representation of the safe paths. The new characterization additionally allows us to understand the space bounds of various representations of all safe paths, where we match the existing lower bounds with worst case upper bounds.

In the future, it would be interesting to see an optimal output-sensitive algorithm for computing even the optimal representation of the safe paths (dropping the $O(\log n)$ factor). It would also be interesting to see if similar properties or algorithms can be used to solve related problems as path covers, or path decomposition for general graphs.

References

- 1 Jasmijn A. Baaijens, Bastiaan Van der Roest, Johannes Köster, Leen Stougie, and Alexander Schönhuth. Full-length de novo viral quasispecies assembly through variation graph construction. *Bioinform.*, 35(24):5086–5094, 2019. doi:10.1093/bioinformatics/btz443.
- 2 Jasmijn A. Baaijens, Leen Stougie, and Alexander Schönhuth. Strain-aware assembly of genomes from mixed samples using flow variation graphs. In *Research in Computational Molecular Biology - 24th Annual International Conference, RECOMB 2020, Padua, Italy, May 10-13, 2020, Proceedings*, pages 221–222, 2020.
- 3 Georg Baier, Ekkehard Köhler, and Martin Skutella. The k-splittable flow problem. *Algorithmica*, 42(3-4):231–248, 2005. doi:10.1007/s00453-005-1167-9.
- 4 Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.
- 5 Elsa Bernard, Laurent Jacob, Julien Mairal, and Jean-Philippe Vert. Efficient RNA isoform identification and quantification from rna-seq data with network flows. *Bioinform.*, 30(17):2447–2455, 2014. doi:10.1093/bioinformatics/btu317.
- 6 Rami Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, and Danny Raz. On the effect of forwarding table size on sdn network utilization. In *IEEE INFOCOM 2014-IEEE conference on computer communications*, pages 1734–1742. IEEE, 2014.
- 7 D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, USA, 2010.
- 8 Thomas Gatter and Peter F Stadler. Ryūtō: network-flow based transcriptome reconstruction. *BMC bioinformatics*, 20(1):190, 2019. doi:10.1186/s12859-019-2786-5.
- 9 Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michal Segalov. How to split a flow? In *2012 Proceedings IEEE INFOCOM*, pages 828–836. IEEE, 2012.
- 10 Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 15–26, 2013.
- 11 Shahbaz Khan, Milla Kortelainen, Manuel Cáceres, Lucia Williams, and Alexandru I. Tomescu. Safety and completeness in flow decompositions for RNA assembly. In *26th Annual International Conference, RECOMB 2022, San Diego, CA, USA, May 22-25, 2022*, pages 177–192, 2022.
- 12 Kyle Kloster, Philipp Kuinke, Michael P O’Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. A practical fpt algorithm for flow decomposition and transcript assembly. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–86. SIAM, 2018.
- 13 Cong Ma, Hongyu Zheng, and Carl Kingsford. Exact transcript quantification over splice graphs. In *20th International Workshop on Algorithms in Bioinformatics, WABI 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, pages 12:1–12:18, 2020.
- 14 Veli Mäkinen, Djamel Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing*. Cambridge University Press, 2015. doi:10.1017/CB09781139940023.
- 15 Brendan Mumey, Samareh Shahmohammadi, Kathryn McManus, and Sean Yaw. Parity balancing path flow decomposition and routing. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2015.

- 16 Jan Peter Ohst. *On the Construction of Optimal Paths from Flows and the Analysis of Evacuation Scenarios*. PhD thesis, University of Koblenz and Landau, Germany, 2015.
- 17 Nils Olsen, Natalia Kliewer, and Lena Wolbeck. A study on flow decomposition methods for scheduling of electric buses in public transport based on aggregated time–space network models. *Central European Journal of Operations Research*, 2020. doi:10.1007/s10100-020-00705-6.
- 18 Mihaela Perteau, Geo M Perteau, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. Stringtie enables improved reconstruction of a transcriptome from rna-seq reads. *Nature biotechnology*, 33(3):290–295, 2015. doi:10.1038/nbt.3122.
- 19 Krzysztof Pieńkosz and Kamil Kołtyś. Integral flow decomposition with minimum longest path length. *European Journal of Operational Research*, 247(2):414–420, 2015. doi:10.1016/j.ejor.2015.06.012.
- 20 Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(2):658–670, 2017.
- 21 Vorapong Suppakitpaisarn. An approximation algorithm for multiroute flow decomposition. *Electronic Notes in Discrete Mathematics*, 52:367–374, 2016. INOC 2015 – 7th International Network Optimization Conference.
- 22 Alexandru I. Tomescu, Travis Gagie, Alexandru Popa, Romeo Rizzi, Anna Kuosmanen, and Veli Mäkinen. Explaining a weighted DAG with few paths for solving genome-guided multi-assembly. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 12(6):1345–1354, 2015. doi:10.1109/TCBB.2015.2418753.
- 23 Alexandru I Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with rna-seq. *BMC bioinformatics*, 14(S5):S15, 2013. doi:10.1186/1471-2105-14-S5-S15.
- 24 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *Journal of Computational Biology*, 24(6):590–602, 2017. Preliminary version appeared in RECOMB 2016.
- 25 Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008. doi:10.1016/j.ejor.2006.05.043.
- 26 Zhong Wang, Mark Gerstein, and Michael Snyder. RNA-Seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, 2009. doi:10.1038/nrg2484.
- 27 Lucia Williams, Gillian Reynolds, and Brendan Mumeay. Rna transcript assembly using inexact flows. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1907–1914. IEEE, 2019.
- 28 Hongyu Zheng, Cong Ma, and Carl Kingsford. Deriving ranges of optimal estimated transcript expression due to nonidentifiability. *J. Comput. Biol.*, 29(2):121–139, 2022. doi:10.1089/cmb.2021.0444.

Scheduling Kernels via Configuration LP

Dušan Knop ✉

Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic

Martin Koutecký ✉

Computer Science Institute of Charles University, Prague, Czech Republic

Abstract

Makespan minimization (on parallel identical or unrelated machines) is arguably the most natural and studied scheduling problem. A common approach in practical algorithm design is to reduce the size of a given instance by a fast preprocessing step while being able to recover key information even after this reduction. This notion is formally studied as kernelization (or simply, kernel) – a polynomial time procedure which yields an equivalent instance whose size is bounded in terms of some given parameter. It follows from known results that makespan minimization parameterized by the longest job processing time p_{\max} has a kernelization yielding a reduced instance whose size is exponential in p_{\max} . Can this be reduced to polynomial in p_{\max} ?

We answer this affirmatively not only for makespan minimization, but also for the (more complicated) objective of minimizing the weighted sum of completion times, also in the setting of unrelated machines when the number of machine kinds is a parameter.

Our algorithm first solves the Configuration LP and based on its solution constructs a solution of an intermediate problem, called huge N -fold integer programming. This solution is further reduced in size by a series of steps, until its encoding length is polynomial in the parameters. Then, we show that huge N -fold IP is in NP, which implies that there is a polynomial reduction back to our scheduling problem, yielding a kernel.

Our technique is highly novel in the context of kernelization, and our structural theorem about the Configuration LP is of independent interest. Moreover, we show a polynomial kernel for huge N -fold IP conditional on whether the so-called separation subproblem can be solved in polynomial time. Considering that integer programming does not admit polynomial kernels except for quite restricted cases, our “conditional kernel” provides new insight.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Integer programming

Keywords and phrases Scheduling, Kernelization

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.73

Related Version *Full Version*: <https://arxiv.org/abs/2003.02187>

Funding *Dušan Knop*: Supported by the OP VVV MEYS funded project “Research Center for Informatics” (nr. CZ.02.1.01/0.0/0.0/16_019/0000765).

Martin Koutecký: Partially supported by Charles University project UNCE/SCI/004, by the Czech Science Foundation (GA ČR) project 19-27871X, and by the Israel Science Foundation grant 308/18.

Acknowledgements Authors wish to thank the Lorentz Center for making it possible to organize the workshop Scheduling Meets Fixed-Parameter Tractability, which output resulted in this paper. The contribution of the Lorentz Center in stimulating suggestions, giving feedback and taking care of all practicalities, helped us to focus on our research and to organize a meeting of high scientific quality. Furthermore, the authors thank Stefan Kratsch for bringing Proposition 5 to their attention.



© Dušan Knop and Martin Koutecký;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 73; pp. 73:1–73:15
Leibniz International Proceedings in Informatics



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

1 Introduction

Kernelization, data reduction, or preprocessing: all of these refer to the goal of simplifying and reducing (the size of) the input in order to speed up computation of challenging tasks. Many heuristic techniques are applied in practice, however, we seek a theoretical understanding in the form of a procedure with guaranteed bounds on the sizes of the reduced data. We use the notion of kernelization from parameterized complexity (cf. [39, 9]), where along with an input instance I we get a positive integer k expressing the *parameter value*, which may be the size of the sought solution or some structural limitation of the input. A kernel is an algorithm running in time $\text{poly}(|I| + k)$ which returns a reduced instance I' of the same problem of size bounded in terms of k ; we sometimes also refer to I' as the kernel.

It is well known [6] that a problem admits a kernel if and only if it has an algorithm running in time $f(k) \text{poly}(|I|)$ for some computable function f (i.e., if it is fixed-parameter tractable, or FPT, parameterized by k). The “catch” is that this kernel may be very large (exponential or worse) in terms of k , while for many problems, kernels of size polynomial in k are known. This raises a fundamental question for any FPT problem: does it have a polynomial kernel? Answering this question typically provides deep insights into a problem and the structure of its solutions.

Parameterized complexity has historically focused primarily on graph problems, but it has been increasingly branching out into other areas. Kernelization, as arguably the most important subfield of parameterized complexity (cf. a recent monograph [15]), follows suit. Scheduling is a fundamental area in combinatorial optimization, with results from parameterized complexity going back to 1995 [3]. Probably the most central problem in scheduling is makespan minimization on identical machines, denoted as $P||C_{\max}$, which we shall define soon. It took until the seminal paper of Goemans and Rothvoß [20] to get an FPT algorithm for $P||C_{\max}$ parameterized by the number of job types (hence also by the largest job). Yet, the existence of a polynomial kernel for $P||C_{\max}$ remained open, despite being raised by Mnich and Wiese in 2013 [38] and reiterated by van Bevern¹. Here, we give an affirmative answer for this problem:

► **Corollary 1.** *There is a polynomial kernel for $P||C_{\max}$ when parameterized by the longest processing time p_{\max} .*

Let us now introduce and define the scheduling problems $P||C_{\max}$ and $P||\sum w_j C_j$. There are n jobs and m identical machines, and the goal is to find a schedule minimizing an objective. For each job $j \in [n]$, a *processing time* $p_j \in \mathbb{N}$ is given and a *weight* w_j are given; in the case of $P||C_{\max}$ the weights play no role and can be assumed to be all zero. A *schedule* is a mapping which to each job $j \in [n]$ assigns some machine $i \in [m]$ and a closed interval of length p_j , such that the intervals assigned to each machine do not overlap except for their endpoints. For each job $j \in [n]$, denote by C_j its *completion time*, which is the time when it finishes, i.e., the right end point of the interval assigned to j in the schedule. In the *makespan minimization* (C_{\max}) problem, the goal is to find a schedule minimizing the time when the last job finishes $C_{\max} = \max_{j \in [n]} C_j$, called the *makespan*. In the *minimization of sum of weighted completion times* ($\sum w_j C_j$), the goal is to minimize $\sum w_j C_j$. (In the rest of the paper we formally deal with *decision* versions of these problems, where the task is to decide whether there exists a schedule with objective value at most k . This is a necessary approach when speaking of kernels and complexity classes like NP and FPT.)

¹ The question was asked at the workshop “Scheduling & FPT” at the Lorentz Center, Leiden, in February 2019, as a part of the opening talk for the open problem session.

In fact, our techniques imply results stronger in three ways, where we handle:

1. the much more complicated $\sum w_j C_j$ objective function involving possibly large job weights,
2. the unrelated machines setting (denoted $R||C_{\max}$ and $R||\sum w_j C_j$), and
3. allowing the number of jobs and machines to be very large, known as the *high-multiplicity* setting.

For this, we need further notation to allow for different kinds of machines. For each machine $i \in [m]$ and job $j \in [n]$, a *processing time* $p_j^i \in \mathbb{N}$ is given. For a given scheduling instance, say that two jobs $j, j' \in [n]$ are of the same *type* if $p_j^i = p_{j'}^i$, for all $i \in [m]$ and $w_j = w_{j'}$, and say that two machines $i, i' \in [m]$ are of the same *kind* if $p_j^i = p_j^{i'}$ for all jobs $j \in [n]$. We denote by $\tau \in \mathbb{N}$ and $\kappa \in \mathbb{N}$ the number of job types and machine kinds, respectively, call this type of encoding the *high-multiplicity* encoding, and denote the corresponding problems $R|HM|C_{\max}$ and $R|HM|\sum w_j C_j$.

Our approach is indirect: taking an instance I of scheduling, we produce a small equivalent instance I' of a the so-called *huge N -fold integer programming* problem with a quadratic objective function (see more details below). This is known as *compression*, i.e., a polynomial time algorithm producing from I a small equivalent instance of a different problem:

► **Theorem 2.** *The problems $R|HM|C_{\max}$ and $R|HM|\sum w_j C_j$ parameterized by the number of job types τ , the longest processing time p_{\max} , and the number of machine kinds κ admit a polynomial compression to quadratic huge N -fold IP parameterized by the number of block types $\bar{\tau}$, the block dimension t , and the largest coefficient $\|E\|_{\infty}$.*

If we can then find a polynomial reduction from quadratic huge N -fold IP to our scheduling problems, we are finished. For this, it suffices to show NP membership, as we do in Lemma 13.

Configuration LP. Besides giving polynomial kernels for some of the most fundamental scheduling problems, we wish to highlight the technique behind this result, because it is quite unlike most techniques used in kernelization and is of independent interest. Our algorithm essentially works by solving the natural Configuration LP of $P||C_{\max}$ (and other problems), which can be done in polynomial time when p_{\max} is polynomially bounded, and then using powerful structural insights to reduce the scheduling instance based on the Configuration LP solution. The Configuration LP is a fundamental tool in combinatorial optimization which goes back to the work of Gilmore and Gomory in 1961 [18]. It is known to provide high-quality results in practice, in fact, the “modified integer round-up property (MIRUP)” conjecture states that the natural Bin Packing Configuration LP always attains a value x such that the integer optimum is at most $\lfloor x \rfloor + 1$ [40]. The famous approximation algorithm of Karmarkar and Karp [28] for Bin Packing is based on rounding the Configuration LP, and many other results in approximation use the Configuration LP for their respective problems as the starting point.

In spite of this centrality and vast importance of the Configuration LP, there are only few structural results providing deeper insight. Perhaps the most notable is the work of Goemans and Rothvoß [20] and later Jansen and Klein [26] who show that there is a certain set of “fundamental configurations” such that in any integer optimum, all but few machines (bins, etc.) will use these fundamental configurations. Our result is based around a theorem which shows a similar yet orthogonal result and can be informally stated as follows:

► **Theorem 3** (informal; see the full version). *There is an optimum of the Configuration IP where all but few configuration are those discovered by the Configuration LP, and the remaining configurations are not far from those discovered by the Configuration LP.*

We note that our result, unlike the ones mentioned above [20, 26], also applies to arbitrary separable convex functions. This has a fundamental reason: the idea behind both previous results is to shift weight from the inside of a polytope to its vertices without affecting the objective value, which only works for linear objectives.

Huge N -fold IP. Finally, we highlight that the engine behind our kernels, a conditional kernel for the so-called quadratic huge N -fold IP, is of independent interest. Integer programming is a central problem in combinatorial optimization. Its parameterized complexity has been recently intensely studied [11, 31, 33, 10, 7]. However, it turns out that integer programs cannot be kernelized in all but the most restricted cases [34, 35, 26]. We give a positive result about a class of block-structured succinctly encoded IPs with a quadratic objective function, so-called quadratic huge N -fold IPs, which was used to obtain many interesting FPT results [29, 31, 4, 17, 32, 5]. However, our result is conditional on having a polynomial algorithm for the so-called *separation subproblem* of the Configuration LP of the quadratic huge N -fold IP, so there is a price to pay for the generality of this fragment of IP. The separation subproblem is to optimize a certain objective function (which varies) over the set of configurations. In the cases considered here, we show that this corresponds to (somewhat involved) variations of the knapsack problem with polynomially bounded numbers; in other problems expressible as n -fold IP, the separation subproblem corresponds to a known hard problem. Informally, our result reads as follows:

► **Theorem 4** (informal; see the full version). *If the separation subproblem can be solved in polynomial time, then quadratic huge N -fold IP has a polynomial kernel parameterized by the block dimensions, the number of block types, and the largest coefficient.*

Because huge N -fold IP is essentially equivalent to the Configuration IP, the kernel of Theorem 3 can be viewed as a validation of the industry common wisdom that column generation works really well. Another view is that methods from mathematical programming, so far underrepresented in kernelization, deserve more attention.

One aspect of the algorithm above is reducing the quadratic objective function. The standard approach, also used in kernelization of weighted problems [13, 8, 2, 19, 42, 41, 21] is to use a theorem of Frank and Tardos [16] which “kernelizes” a linear objective function if the dimension is a parameter. However, we deal with

1. a quadratic convex (non-linear) function,
2. over a space of large dimension.

We are able to overcome these obstacles by a series of steps which first “linearize” the objective, then “aggregate” variables of the same type, hence shrinking the dimension, then reduce the objective using the algorithm of Frank and Tardos, and then we carefully reverse this process (see the full version for more details). This result has applications beyond this work: for example, the currently fastest strongly FPT algorithm for $R||\sum w_j C_j$ (i.e., an algorithm whose number of arithmetic operations does not depend on the weights w_j) has dependence of $m^2 \text{poly log}(m)$ on the number of machines m ; applying our new result instead of [11, Corollary 69] reduces this dependence to $m \text{poly log}(m)$.

Other Applications. Theorem 4 can be used to obtain kernels for other problems which can be modeled as huge N -fold IP. First, we may also optimize the ℓ_p norms of times when each machine finishes, a problem known as $R|HM|\ell_p$. Our results (Corollary 11) show that also in this setting the separation problem can be solved quickly. Second, the $P||C_{\max}$ problem is identical to Bin Packing (in their decision form), so our kernel also gives a kernel for

Bin Packing parameterized by the largest item size. Moreover, also the Bin Packing with Cardinality Constraints problem has a huge N -fold IP model [30, Lemma 54] for which Corollary 11 indicates that the separation subproblem can be solved quickly. Third, Knop et al. [30] give a huge N -fold IP model for the SURFING problem, in which many “surfers” make demands on few different “services” provided by few “servers”, where each surfer may have different costs of getting a service from a server; one may think of internet streaming with different content types, providers, and pricing schemes for different customer types. The separation problem there is polynomially solvable for an interesting reason: its constraint matrix is totally unimodular because it is the incidence matrix of the complete bipartite graph. Thus, Theorem 4 gives polynomial kernels for all of the problems above with the given parameters.

Related Work – Scheduling. Let us finally review related results in the intersection of parameterized complexity and scheduling; for a more comprehensive survey of parameterized results in scheduling see, e.g., [37]. First, to the best of our knowledge, the first to study scheduling problems from the perspective of multivariate complexity were Bodlaender and Fellows [3]. Fellows and McCartin [14] study scheduling on single machine of unit length jobs with (many) different release times and due dates. Single machine scheduling where two agents compete to schedule their private jobs is investigated by Hermelin et al. [22]. There are few other result [43, 27, 24, 23] focused on identifying tractable scenarios for various scheduling paradigms (such as flow-shop scheduling or e.g. structural limitations of the job–machine assignment).

Paper Organization. We begin with preliminaries (Section 2) which, besides introducing necessary notation and definitions, also states several results which we use later in the proofs of our results. Section 3 discusses how our scheduling problems are modeled as huge N -fold IP (3.1) and how to solve the ConfLP quickly for those models (3.2 and 3.3). Finally, in the full version, we show our “conditional kernel” for quadratic huge N -fold IP, which first reduces all parts of the instance except for the objective function, and finally deals with the objective. In the full version, we conclude with a short section giving an interpretation of our algorithm for $P||C_{\max}$ and $R||C_{\max}$. A word of caution: it is at first tempting to think that most of the machinery of our algorithm is not necessary for our simplest considered problem $P||C_{\max}$; however, as we discuss in the full version in detail, while some minor simplifications are possible, the only truly avoidable step is the objective reduction which is needed for the $\sum w_j C_j$ objective.

2 Preliminaries

We consider zero to be a natural number, i.e., $0 \in \mathbb{N}$. We write vectors in boldface (e.g., \mathbf{x}, \mathbf{y}) and their entries in normal font (e.g., the i -th entry of a vector \mathbf{x} is x_i). For positive integers $m \leq n$ we set $[m, n] := \{m, \dots, n\}$ and $[n] := [1, n]$, and we extend this notation for vectors: for $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^n$ with $\mathbf{l} \leq \mathbf{u}$, $[\mathbf{l}, \mathbf{u}] := \{\mathbf{x} \in \mathbb{Z}^n \mid \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$ (where we compare component-wise). For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{z} = \max\{\mathbf{x}, \mathbf{y}\}$ is defined coordinate-wise, i.e., $z_i = \max\{x_i, y_i\}$ for all $i \in [n]$, and similarly for $\min\{\mathbf{x}, \mathbf{y}\}$.

If A is a matrix, $A_{i,j}$ denotes the j -th coordinate of the i -th row, $A_{i,\bullet}$ denotes the i -th row and $A_{\bullet,j}$ denotes the j -th column. We use $\log := \log_2$, i.e., all our logarithms are base 2. For an integer $a \in \mathbb{Z}$, we denote by $\langle a \rangle := 1 + \lceil \log(|a| + 1) \rceil$ the binary encoding length of a ; we extend this notation to vectors, matrices, and tuples of these objects. For example,

$\langle A, \mathbf{b} \rangle = \langle A \rangle + \langle \mathbf{b} \rangle$, and $\langle A \rangle = \sum_{i,j} \langle A_{i,j} \rangle$.² For a function $f: \mathbb{Z}^n \rightarrow \mathbb{Z}$ and two vectors $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^n$, we define $f_{\max}^{[\mathbf{l}, \mathbf{u}]} := \max_{\mathbf{x} \in [\mathbf{l}, \mathbf{u}]} |f(\mathbf{x})|$; if $[\mathbf{l}, \mathbf{u}]$ is clear from the context we omit it and write just f_{\max} .

► **Proposition 5** ([15, Theorem 1.6]). *Let $(Q, \kappa), (R, \lambda)$ be parameterized problems such that Q is NP-hard and R is in NP. If (Q, κ) admits a polynomial compression into (R, λ) , then it admits a polynomial kernel.*

The above observation is useful when dealing with NP-hard problems. The proof simply follows by pipelining the assumed polynomial compression with a polynomial time (Karp) reduction from R to Q .

2.1 Scheduling Notation

Overloading the convention slightly, for each $i \in [\kappa]$ and $j \in [\tau]$, denote by p_j^i the processing time of a job of type j on a machine of kind i , by w_j the weight of a job of type j , by n_j the number of jobs of type j , by m_i the number of machines of kind i , and denote $\mathbf{n} = (n_1, \dots, n_\tau)$, $\mathbf{m} = (m_1, \dots, m_\kappa)$, $\mathbf{p} = (p_1^1, \dots, p_\tau^1, p_1^2, \dots, p_\tau^\kappa)$, $\mathbf{w} := (w_1, \dots, w_\tau)$, $p_{\max} := \|\mathbf{p}\|_\infty$, and $w_{\max} := \|\mathbf{w}\|_\infty$. We denote the high multiplicity versions of the previously defined problems $R|HM|C_{\max}$ and $R|HM|\sum w_j C_j$.

For an instance I of $R|C_{\max}$ or $R|\sum w_j C_j$, we define its size as $\langle I \rangle := \sum_{i=1}^\kappa \sum_{j=1}^\tau \langle p_j^i, w_j \rangle$, whereas for an instance I of $P|HM|C_{\max}$ or $P|HM|\sum w_j C_j$ we define its size as $\langle I \rangle = \langle \mathbf{n}, \mathbf{m}, \mathbf{p}, \mathbf{w} \rangle$. Note that the difference in encoding actually leads to different problems: for example, an instance of $R|HM|C_{\max}$ with 2^k jobs with maximum processing time p_{\max} can be encoded with $\mathcal{O}(k\tau\kappa \log p_{\max})$ bits while an equivalent instance of $R|C_{\max}$ needs $\Omega(2^k \log p_{\max})$ bits, which is exponentially more if $\tau, \kappa \in k^{\mathcal{O}(1)}$. The membership of high-multiplicity scheduling problems in NP was open for some time, because it is not obvious whether a compactly encoded instance also has an optimal solution with a compact encoding. This question was considered by Eisenbrand and Shmonin, and we shall use their result. For a set $X \subseteq \mathbb{Z}^d$ define the *integer cone* of X , denoted $\text{cone}_{\mathbb{N}}(X)$, to be the set $\text{cone}_{\mathbb{N}}(X) := \{ \sum_{\mathbf{x} \in X} \lambda_{\mathbf{x}} \mathbf{x} \mid \lambda \in \mathbb{N}^X \}$, where \mathbb{N}^X is the set of functions mapping $X \rightarrow \mathbb{N}$ viewed as vectors.

► **Proposition 6** (Eisenbrand and Shmonin [12, Theorem 2]). *Let $X \subseteq \mathbb{Z}^d$ be a finite set of integer vectors and let $\mathbf{b} \in \text{cone}_{\mathbb{N}}(X)$. Then there exists a subset $\tilde{X} \subseteq X$ such that $\mathbf{b} \in \text{cone}_{\mathbb{N}}(\tilde{X})$ and the following holds for the cardinality of \tilde{X} :*

1. *if all vectors of X are nonnegative, then $|\tilde{X}| \leq \langle \mathbf{b} \rangle$,*
2. *if $M = \max_{\mathbf{x} \in X} \|\mathbf{x}\|_\infty$, then $|\tilde{X}| \leq 2d(\log 4dM)$.*

One can use Proposition 6 to show that the decision version of $R|HM|C_{\max}$ and $R|HM|\sum w_j C_j$ have short certificates and thus belong to NP. We will later derive the same result as a corollary of the fact that both of these scheduling problems can be encoded as a certain form of integer programming, which we will show to have short certificates as well.

2.2 Conformal Order and Graver Basis

Let $\mathbf{g}, \mathbf{h} \in \mathbb{Z}^n$ be two vectors. We say that \mathbf{g} is *conformal* to \mathbf{h} (we denote it $\mathbf{g} \sqsubseteq \mathbf{h}$) if both $g_i \cdot h_i \geq 0$ and $|g_i| \leq |h_i|$ for all $i \in [n]$. In other words, $\mathbf{g} \sqsubseteq \mathbf{h}$ if they are in the same

² We note that our encoding of integers already contains the delimiter symbol.

orthant (the first condition holds) and \mathbf{g} is component-wise smaller than \mathbf{h} . For a matrix A we define its *Graver basis* (A) to be the set of all \sqsubseteq -minimal vectors in $\text{Ker}(A) \setminus \{\mathbf{0}\}$. We define $g_\infty(A) = \max\{\|\mathbf{g}\|_\infty \mid \mathbf{g} \in (A)\}$ and $g_1(A) = \max\{\|\mathbf{g}\|_1 \mid \mathbf{g} \in (A)\}$.

We say that two functions $f, g: \mathbb{Z}^d \rightarrow \mathbb{Z}$ are *equivalent* on a polyhedron $P \subseteq \mathbb{Z}^d$ if $f(\mathbf{x}) \leq f(\mathbf{y})$ if and only if $g(\mathbf{x}) \leq g(\mathbf{y})$ for all $\mathbf{x}, \mathbf{y} \in P$. Note that if f and g are equivalent on P , then the sets of minimizers of $f(\mathbf{x})$ and $g(\mathbf{x})$ over P coincide.

► **Proposition 7** (Frank and Tardos [16]). *Given a rational vector $\mathbf{w} \in \mathbb{Q}^d$ and an integer M , there is a polynomial algorithm which finds a $\tilde{\mathbf{w}} \in \mathbb{Z}^d$ such that the linear functions $\mathbf{w}\mathbf{x}$ and $\tilde{\mathbf{w}}\mathbf{x}$ are equivalent on $[-M, M]^d$, and $\|\tilde{\mathbf{w}}\|_\infty \leq 2^{\mathcal{O}(d^3)} M^{\mathcal{O}(d^2)}$.*

The *dual graph* $G_D(A) = (V, E)$ of a matrix $A \in \mathbb{Z}^{m \times n}$ has $V = [m]$ and $\{i, j\} \in E$ if rows i and j contain a non-zero at a common coordinate $k \in [n]$. The dual treewidth $\text{tw}_D(A)$ of A is $\text{tw}(G_D(A))$. We do not define treewidth here, but we point out that $\text{tw}(T) = 1$ for every tree T .

► **Proposition 8** (Eisenbrand et al. [11, Theorem 98]). *An IP with a constraint matrix A can be solved in time $(\|A\|_\infty g_1(A))^{\mathcal{O}(\text{tw}_D(A))} \text{poly}(n, L)$, where n is the dimension of the IP and L is the length of the input.*

► **Proposition 9** (Eisenbrand et al. [11, Lemma 25]). *For an integer matrix $A \in \mathbb{Z}^{m \times n}$, we have $g_1(A) \leq (2\|A\|_\infty m + 1)^m$.*

2.3 N-fold Integer Programming

The INTEGER PROGRAMMING problem is to solve:

$$\min f(\mathbf{x}) : A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^n, \tag{IP}$$

where $f: \mathbb{R}^n \rightarrow \mathbb{R}$, $A \in \mathbb{Z}^{m \times n}$, $\mathbf{b} \in \mathbb{Z}^m$, and $\mathbf{l}, \mathbf{u} \in (\mathbb{Z} \cup \{\pm\infty\})^n$.

A *generalized N-fold IP matrix* is defined as

$$E^{(N)} = \begin{pmatrix} E_1^1 & E_1^2 & \cdots & E_1^N \\ E_2^1 & 0 & \cdots & 0 \\ 0 & E_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & E_2^N \end{pmatrix}. \tag{1}$$

Here, $r, s, t, N \in \mathbb{N}$, $E^{(N)}$ is an $(r + Ns) \times Nt$ -matrix, and $E_1^i \in \mathbb{Z}^{r \times t}$ and $E_2^i \in \mathbb{Z}^{s \times t}$ for all $i \in [N]$, are integer matrices. Problem (IP) with $A = E^{(N)}$ is known as *generalized N-fold integer programming* (generalized N-fold IP). “Regular” N-fold IP is the problem where $E_1^i = E_1^j$ and $E_2^i = E_2^j$ for all $i, j \in [N]$. Recent work indicates that the majority of techniques applicable to “regular” N-fold IP also applies to generalized N-fold IP [11].

The structure of $E^{(N)}$ allows us to divide any Nt -dimensional object, such as the variables of \mathbf{x} , bounds \mathbf{l}, \mathbf{u} , or the objective f , into N bricks of size t , e.g. $\mathbf{x} = (\mathbf{x}^1, \dots, \mathbf{x}^N)$. We use subscripts to index within a brick and superscripts to denote the index of the brick, i.e., x_j^i is the j -th variable of the i -th brick with $j \in [t]$ and $i \in [N]$. We call a brick *integral* if all of its coordinates are integral, and *fractional* otherwise.

Huge N -fold IP. The *huge N -fold IP* problem is an extension of generalized N -fold IP to the high-multiplicity scenario, where blocks come in types and are encoded succinctly by type multiplicities. This means there could be an *exponential* number of bricks in an instance with a polynomial encoding size. The input to the huge N -fold IP problem with $\bar{\tau}$ types of blocks is defined by matrices $E_1^i \in \mathbb{Z}^{r \times t}$ and $E_2^i \in \mathbb{Z}^{s \times t}$, $i \in [\bar{\tau}]$, vectors $\mathbf{l}^1, \dots, \mathbf{l}^{\bar{\tau}}, \mathbf{u}^1, \dots, \mathbf{u}^{\bar{\tau}} \in \mathbb{Z}^t$, $\mathbf{b}^0 \in \mathbb{Z}^r$, $\mathbf{b}^1, \dots, \mathbf{b}^{\bar{\tau}} \in \mathbb{Z}^s$, functions $f^1, \dots, f^{\bar{\tau}}: \mathbb{R}^t \rightarrow \mathbb{R}$ satisfying $\forall i \in [\bar{\tau}], \forall \mathbf{x} \in \mathbb{Z}^t$ we have $f^i(\mathbf{x}) \in \mathbb{Z}$ and given by evaluation oracles, and integers $\mu^1, \dots, \mu^{\bar{\tau}} \in \mathbb{N}$ such that $\sum_{i=1}^{\bar{\tau}} \mu^i = N$. We say that a brick is of type i if its lower and upper bounds are \mathbf{l}^i and \mathbf{u}^i , its right hand side is \mathbf{b}^i , its objective is f^i , and the matrices appearing at the corresponding coordinates are E_1^i and E_2^i . Denote by T_i the indices of bricks of type i , and note $|T_i| = \mu_i$ and $|\bigcup_{i \in [\bar{\tau}]} T_i| = N$. The task is to solve (IP) with a matrix $E^{(N)}$ which has μ^i blocks of type i for each i . Knop et al. [30] have shown a fast algorithm solving huge n -fold IP. The main idea of their approach is to prove a powerful proximity theorem showing how one can drastically reduce the size of the input instance given that one can solve a corresponding configuration LP (which we shall formally define later). We will build on this approach here. When f^i are restricted to be separable quadratic (and convex) for all $i \in [\bar{\tau}]$, we call the problem *quadratic huge N -fold IP*.

2.4 Configuration LP of Huge N -fold IP

Let a huge N -fold IP instance with $\bar{\tau}$ types be fixed. Recall that μ^i denotes the number of blocks of type i , and let $\boldsymbol{\mu} = (\mu^1, \dots, \mu^{\bar{\tau}})$. We define for each $i \in [\bar{\tau}]$ the set of configurations of type i as

$$\mathcal{C}^i = \{ \mathbf{c} \in \mathbb{Z}^t \mid E_2^i \mathbf{c} = \mathbf{b}^i, \mathbf{l}^i \leq \mathbf{c} \leq \mathbf{u}^i \} .$$

Here we are interested in four instances of convex programming (CP) and convex integer programming (IP) related to huge N -fold IP. First, we have the *Huge IP*

$$\min f(\mathbf{x}) : E^{(N)} \mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^{Nt} . \quad (\text{HugeIP})$$

Then, there is the *Configuration LP* of (HugeIP),

$$\begin{aligned} \min \mathbf{v}\mathbf{y} &= \sum_{i=1}^{\bar{\tau}} \sum_{\mathbf{c} \in \mathcal{C}^i} f^i(\mathbf{c}) \cdot y(i, \mathbf{c}) & (2) \\ \sum_{i=1}^{\bar{\tau}} E_1^i \sum_{\mathbf{c} \in \mathcal{C}^i} \mathbf{c} y(i, \mathbf{c}) &= \mathbf{b}^0 \\ \sum_{\mathbf{c} \in \mathcal{C}^i} y(i, \mathbf{c}) &= \mu^i & \forall i \in [\bar{\tau}] \\ \mathbf{y} &\geq \mathbf{0} . & (3) \end{aligned}$$

Let B be its constraint matrix and $\mathbf{d} = (\mathbf{b}^0, \boldsymbol{\mu})$ be the right hand side and shorten (2)-(3) to

$$\min \mathbf{v}\mathbf{y} : B\mathbf{y} = \mathbf{d}, \mathbf{y} \geq \mathbf{0} . \quad (\text{ConfLP})$$

Finally, by observing that $B\mathbf{y} = \mathbf{d}$ implies $y(i, \mathbf{c}) \leq \|\boldsymbol{\mu}\|_\infty$ for all $i \in [\bar{\tau}], \mathbf{c} \in \mathcal{C}^i$, defining $C = \sum_{i \in [\bar{\tau}]} |\mathcal{C}^i|$, leads to the *Configuration ILP*,

$$\min \mathbf{v}\mathbf{y} : B\mathbf{y} = \mathbf{d}, \mathbf{0} \leq \mathbf{y} \leq (\|\boldsymbol{\mu}\|_\infty, \dots, \|\boldsymbol{\mu}\|_\infty), \mathbf{y} \in \mathbb{N}^C . \quad (\text{ConfILP})$$

The classical way to solve (ConfLP) is by solving its dual using the ellipsoid method and then restricting (ConfLP) to the columns corresponding to the rows encountered while solving the dual, a technique known as column generation. The Dual LP of (ConfLP) in variables $\alpha \in \mathbb{R}^r$, $\beta \in \mathbb{R}^{\bar{\tau}}$ is:

$$\begin{aligned} \max \quad & \mathbf{b}^0 \alpha + \sum_{i=1}^{\bar{\tau}} \mu^i \beta^i \\ \text{s.t.} \quad & (\alpha E_1^i) \mathbf{c} - f^i(\mathbf{c}) \leq -\beta^i \quad \forall i \in [\bar{\tau}], \forall \mathbf{c} \in \mathcal{C}^i \end{aligned} \quad (4)$$

To verify feasibility of (α, β) for $i \in [\bar{\tau}]$, we need to maximize the left-hand side of (4) over all $\mathbf{c} \in \mathcal{C}^i$ and check if it is at most $-\beta^i$. This corresponds to solving the following *separation problem*: find integer variables \mathbf{c} which for a given vector (α, β) solve

$$\min f^i(\mathbf{c}) - (\alpha E_1^i) \mathbf{c} : E_2^i \mathbf{c} = \mathbf{b}^i, \mathbf{l}^i \leq \mathbf{c} \leq \mathbf{u}^i, \mathbf{c} \in \mathbb{Z}^t. \quad (\text{sep-IP})$$

Denote by $\text{sep}(\mathbf{l}^i, \mathbf{u}^i, f_{\max}^i, E_1^i, E_2^i)$ the time needed to solve (sep-IP).

► **Lemma 10** (Knop et al. [30, Lemma 12]). *An optimal solution \mathbf{y}^* of (ConfLP) with $|\text{supp}(\mathbf{y}^*)| \leq r + \bar{\tau}$ can be found in $(rt\bar{\tau} \langle f_{\max}, \mathbf{l}, \mathbf{u}, \mathbf{b}, \boldsymbol{\mu} \rangle)^{\mathcal{O}(1)} \cdot \max_{i \in [\bar{\tau}]} \text{sep}(\mathbf{l}^i, \mathbf{u}^i, f_{\max}^i, E_1^i, E_2^i)$ time.*

Since (sep-IP) is an IP, it can be solved using Proposition 8 in time $g_1(E_2^i)^{\text{tw}_D(E_2^i)} \cdot \text{poly}(\langle \mathbf{l}^i, \mathbf{u}^i, \mathbf{b}^i, \|E_1^i\|_{\infty} f_{\max} \rangle, t, \bar{\tau})$. Hence, together with Lemma 10, we get the following corollary:

► **Corollary 11.** *An optimal solution \mathbf{y}^* of (ConfLP) with $|\text{supp}(\mathbf{y}^*)| \leq r + \bar{\tau}$ can be found in time $(rt\bar{\tau} \langle f_{\max}, \mathbf{l}, \mathbf{u}, \mathbf{b}, \boldsymbol{\mu} \rangle)^{\mathcal{O}(1)} \cdot \max_{i \in [\bar{\tau}]} g_1(E_2^i)^{\text{tw}_D(E_2^i)}$.*

We later show how that for our formulations of $R|HM|C_{\max}$ and $R|HM|\sum w_j C_j$, indeed $g_1(E_2^i)$ is polynomial in τ, p_{\max} , and $\text{tw}_D(E_2^i) = 1$, hence the (ConfLP) optimum can be found in polynomial time.

3 Compressing High Multiplicity Scheduling to Quadratic N -fold IP

In this section we are going to prove Theorem 2. To that end, we use the following assumption, mainly to simplify notation.

► **Remark 12.** From here on, we assume $\tau \geq \|\mathbf{p}\|_{\infty}$, i.e., there is a job type for each possible job length $\{1, 2, \dots, \|\mathbf{p}\|_{\infty}\}$. This is without loss of generality in our regime since both quantities are parameters.

► **Theorem 2 (repeated).** *The problems $R|HM|C_{\max}$ and $R|HM|\sum w_j C_j$ parameterized by the number of job types τ , the longest processing time p_{\max} , and the number of machine kinds κ admit a polynomial compression to quadratic huge N -fold IP parameterized by the number of block types $\bar{\tau}$, the block dimension t , and the largest coefficient $\|E\|_{\infty}$.*

Recall that in order to use Theorem 2 to provide *kernels* for selected scheduling problems (which are NP-hard) we want to utilize Proposition 5. Thus, we have to show that the “target problem” quadratic huge N -fold IP is in NP.

► **Lemma 13.** *The decision version of quadratic huge N -fold IP belongs to NP.*

Proof. We will use Proposition 6 to show that there exists an optimum whose number of distinct configurations is polynomial in the input length. Such a solution can then be encoded by giving those configurations together with their multiplicities, and constitutes a polynomial certificate. Recall that (ConflLP) corresponding to the given instance of huge N -fold is

$$\min \mathbf{v}\mathbf{y} : B\mathbf{y} = \mathbf{d}, \mathbf{0} \leq \mathbf{y} \leq (\|\boldsymbol{\mu}\|_\infty, \dots, \|\boldsymbol{\mu}\|_\infty), \mathbf{y} \in \mathbb{N}^C .$$

Let X be the set of columns of the matrix B extended with an additional coordinate which is the coefficient of the objective function \mathbf{v} corresponding to the given column, that is, $\mathbf{v}\mathbf{b}$ for a column \mathbf{b} (i.e., the objective value of configuration \mathbf{b}). Hence $X \subseteq \mathbb{Z}^{r+\bar{\tau}+1}$ and $\|\mathbf{x}\|_\infty \leq \|\mathbf{l}, \mathbf{u}, f_{\max}\|_\infty =: M$ for any $\mathbf{x} \in X$. Applying Proposition 6, part 2, to X , yields that there exists an optimal solution \mathbf{y} of (ConflLP) with $\text{supp}(\mathbf{y}) = \tilde{X}$ satisfying $|\tilde{X}| \leq 2(r + \bar{\tau} + 1) \log(4(r + \bar{\tau} + 1)M)$, hence polynomial in the input length of the original instance. ◀

► **Remark 14.** Clearly Lemma 13 holds for any huge N -fold IP whose objective is restricted by some, not necessarily quadratic, polynomial. Moreover, using the newer technique of Aliev et al. [1] it is likely possible to remove any restrictions on the objective.

Using Theorem 2. Before we move to the proof of Theorem 2 we first derive two simple yet interesting corollaries.

► **Corollary 15.** *The problems $R||C_{\max}$ and $R||\sum w_j C_j$ admit polynomial kernelizations when parameterized by τ, κ, p_{\max} .*

Proof. Let $\text{obj} \in \{C_{\max}, \sum w_j C_j\}$. We describe a polynomial compression from $R||\text{obj}$ to quadratic huge N -fold IP which, by Lemma 13, yields the sought kernel, since $R||\text{obj}$ is NP-hard and huge N -fold with a quadratic objective is in NP.

We first perform the high-multiplicity encoding of the given instance I of $R||\text{obj}$, thus obtaining an instance I_{HM} of $R|HM|\text{obj}$ with the input encoded as $(\mathbf{n}, \mathbf{m}, \mathbf{p}, \mathbf{w})$. Now, we can apply Theorem 2 and obtain an instance $\tilde{I}_{\text{huge } N\text{-fold}}$ equivalent to I_{HM} with size bounded by a polynomial in κ, τ, p_{\max} . ◀

Proof of Corollary 1. This is now trivial, since it suffices to observe that $P||C_{\max}$ is a special case of $R||C_{\max}$, where there is only a single machine kind (i.e., $\kappa = 1$) and $\tau \leq p_{\max}$ job types. Our claim then follows by Corollary 15 (combined with the fact that $P||C_{\max}$ is NP-hard and $R||C_{\max}$ is in NP). ◀

3.1 Huge n -fold IP Models

Denote by \mathbf{n}_{\max} the τ -dimensional vector whose all entries are $\|\mathbf{n}\|_\infty$. It was shown [29, 30] that $R|HM|C_{\max}$ is modeled as a feasibility instance of huge n -fold IP as follows. Recall that we deal with the decision versions and that k is the upper bound on the value of the objective(s). We set $\mathbf{b}^0 = \mathbf{n}$, the number of block types is $\bar{\tau} = \kappa$, $E_1^i := (I \ \mathbf{0}) \in \mathbb{Z}^{\tau \times (\tau+1)}$, $E_2^i := (\mathbf{p}^i \ 1)$, $\mathbf{l}^i = \mathbf{0}$, $\mathbf{u}^i = (\mathbf{n}, \infty)$, $\mathbf{b}^i = k$, for $i \in [\kappa]$, and the multiplicities of blocks are $\boldsymbol{\mu} = \mathbf{m}$. The meaning is that the first type of constraints expressed by the E_1^i matrices ensures that every job is scheduled somewhere, and the second type of constraints expressed by the E_2^i matrices ensures that every machine finishes in time C_{\max} .

In the model of $R|HM|\sum w_j C_j$, for each machine kind $i \in [\kappa]$, we define \preceq^i to be the ordering of jobs by the w_j/p_j^i ratio non-increasingly, and let $\mathbf{a} = (a_1, a_2, \dots, a_\tau)$ be a reordering of \mathbf{p}^i according to \preceq^i . We let

$$G^i := \begin{pmatrix} a_1 & 0 & 0 & \dots & 0 \\ a_1 & a_2 & 0 & \dots & 0 \\ a_1 & a_2 & a_3 & \dots & 0 \\ \vdots & & & \ddots & \\ a_1 & a_2 & a_3 & \dots & a_\tau \end{pmatrix}, \quad H := -I,$$

with I the $\tau \times \tau$ identity, and define $F^i := (G^i \ H)$ in two steps. Denote by I^{\preceq^i} a matrix obtained from the $\tau \times \tau$ identity matrix by permuting its columns according to \preceq^i . The model is then $\mathbf{b}^0 = \mathbf{n}$, the number of block types is again $\bar{\tau} = \kappa$, for each $i \in [\kappa]$ we have $E_1^i = (I^{\preceq^i} \ \mathbf{0}) \in \mathbb{Z}^{\tau \times 2\tau}$, $E_2^i = \bar{F}^i$, $\mathbf{l}^i = \mathbf{0}$, $\mathbf{u}^i = (\mathbf{n}_{\max}, p_{\max} \tau \mathbf{n}_{\max})$, $\mathbf{b}^i = \mathbf{0}$, f^i is a separable convex quadratic function (whose coefficients are related to the w_j/p_j^i ratios), and again $\boldsymbol{\mu} = \mathbf{m}$. Intuitively, in each brick, the first τ variables represent numbers of jobs of each type on a given machine, and the second τ variables represent the amount of processing time spend by jobs of the first j types with respect to the ordering \preceq^i .

3.2 Solving The Separation Problem Quickly: C_{\max}

The crucial aspect of complexity of (sep-IP) is its constraint matrix E_2^i . For $R|HM|C_{\max}$, this is just the vector $(\mathbf{p}^i, 1)$. Clearly $\text{tw}_D((\mathbf{p}^i, 1)) = 1$ since $G_D((\mathbf{p}^i, 1))$ is a single vertex. By Proposition 9, $g_1((\mathbf{p}^i, 1)) \leq 2\|\mathbf{p}\|_\infty + 1$. Moreover, f_{\max} depends polynomially on $\|\mathbf{n}, \mathbf{m}, \mathbf{p}\|_\infty$. Hence, Corollary 11 states that (ConfLP) of the $R|HM|C_{\max}$ model can be solved in time $(rt\bar{\tau}(f_{\max}, \mathbf{l}, \mathbf{u}, \mathbf{b}, \boldsymbol{\mu}))^{\mathcal{O}(1)} \cdot \max_i g_1(E_2^i)^{\text{tw}_D(E_2^i)} = \text{poly}(p_{\max}, \tau, \log \|\mathbf{n}, \mathbf{m}, \mathbf{p}\|_\infty)$, which is polynomial in the input.

3.3 Solving The Separation Problem Quickly: $\sum w_j C_j$

The situation is substantially more involved in the case of $R|HM|\sum w_j C_j$: in order to apply Corollary 11, we need to again bound $g_1(E_2^i)$ and $\text{tw}_D(E_2^i)$, but the matrix E_2^i is more involved now. Let

$$\bar{G}^i := \begin{pmatrix} a_1 & 0 & \dots & 0 \\ 0 & a_2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & a_\tau \end{pmatrix}, \quad \bar{H} := \begin{pmatrix} -1 & 0 & 0 & \dots & 0 \\ 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & -1 \end{pmatrix},$$

and define $\bar{F}^i = (\bar{G}^i \ \bar{H})$. Now observe that F^i and \bar{F}^i are row-equivalent³. This means that we can replace F^i with \bar{F}^i without changing the meaning of the constraints and without changing the feasible set. But while $\text{tw}_D(F_i) = \tau$ (because it is the clique K_τ), we have

► **Lemma 16** (\star). *For each $i \in [\kappa]$, $\text{tw}_D(\bar{F}^i) = 1$.*

Note that application of Proposition 9 yields $g_1(E_2^i) \leq \mathcal{O}(\tau^\tau)$. This general upper bound, as we shall see, is not sufficient for our purposes, since we need $g_1(E_2^i) \leq \text{poly}(\tau)$. However, we can improve it significantly:

³ Two matrices A, A' are row-equivalent if one can be transformed into the other using elementary row operations.

► **Lemma 17** (Hill-cutting). *We have $g_\infty(F^i), g_\infty(\bar{F}^i) \leq \mathcal{O}(\tau^4)$ and $g_1(F^i), g_1(\bar{F}^i) \leq \mathcal{O}(\tau^5)$ for every $i \in [\kappa]$.*

Proof. Let $F = F^i$ for some $i \in [\kappa]$. Let $(\mathbf{x}, \mathbf{z}) \in \mathbb{Z}^{2\tau}$ be some vector satisfying $F \cdot (\mathbf{x}, \mathbf{z}) = \mathbf{0}$. Our goal now is to show that whenever there exists $k \in [\tau]$ with $|z_k - z_{k-1}| > 2\tau^3 + 1$ (where we define $z_0 := 0$ for convenience), then we can construct a non-zero integral vector $(\mathbf{g}, \mathbf{h}) \in \text{Ker}_{\mathbb{Z}}(F)$ satisfying $(\mathbf{g}, \mathbf{h}) \sqsubseteq (\mathbf{x}, \mathbf{z})$, which shows that $(\mathbf{x}, \mathbf{z}) \notin (A)$. If no such index k exists, it means that $\|\mathbf{x}\|_\infty \leq \mathcal{O}(\tau^3)$ because $z_k - z_{k-1} = a_k x_k$ holds in F (and \bar{F}^i). Moreover, if $|z_k - z_{k-1}| \leq 2\tau^3 + 1$ for all $k \in [\tau]$, then $|z_k| \leq \mathcal{O}(\tau^4)$ for every $k \in [\tau]$, hence $\|(\mathbf{x}, \mathbf{z})\|_\infty \leq \mathcal{O}(\tau^4)$. Note that, since the dimension of (\mathbf{x}, \mathbf{z}) is 2τ , this also implies $\|(\mathbf{x}, \mathbf{z})\|_1 \leq \mathcal{O}(\tau^5)$. Thus we now focus on the case when $\exists k \in [\tau] : |z_k - z_{k-1}| > 2\tau^3 + 1$.

Let us now assume that $(z_k - z_{k-1})$ is positive and $z_k \geq \tau^3$. There are three other possible scenarios: when $(z_k - z_{k-1})$ is positive but $z_k < \tau^3$, or when $(z_k - z_{k-1})$ is negative and $z_k < -\tau^3$ or $z_k \geq -\tau^3$. We will later show that all these situations are symmetric to the one we consider and our arguments carry over easily, hence our assumption is without loss of generality.

▷ **Claim 18** (★). If $(z_k - z_{k-1})$ is positive and $z_k \geq \tau^3$, then there exists nonzero $(\mathbf{g}, \mathbf{h}) \in \text{Ker}_{\mathbb{Z}}(F)$ with $(\mathbf{g}, \mathbf{h}) \sqsubseteq (\mathbf{x}, \mathbf{z})$.

Let us consider the remaining symmetric cases. If $z_k - z_{k-1}$ is negative and $z_k < -\tau^3$, then $(-\mathbf{x}, -\mathbf{z})$ satisfies the original assumption, leading to some $(\mathbf{g}', \mathbf{h}') \sqsubseteq (-\mathbf{x}, -\mathbf{z})$, hence $(-\mathbf{g}', -\mathbf{h}') \sqsubseteq (\mathbf{x}, \mathbf{z})$ and we are done. If $z_k - z_{k-1}$ is negative but $z_k > -\tau^3$, then we would pick the largest index ℓ smaller than k with $x_\ell > \tau$ and continue as before (the symmetry is that now ℓ is to the left of k rather than to its right; that is, the case distinction from the previous paragraph is according to the value of z_1). Lastly, if $z_k - z_{k-1}$ is positive but $z_k < \tau^3$, negating (\mathbf{x}, \mathbf{z}) gives a reduction to the previous case. ◀

Together with the observation from the previous section and using our newly obtained bounds together with Corollary 11, we obtain:

► **Corollary 19.** *Let $I = (\mathbf{n}, \mathbf{m}, \mathbf{p}, \mathbf{w})$ be an instance of $R|HM|C_{\max}$ or $R|HM|\sum w_j C_j$. A (ConfLP) optimum \mathbf{y}^* with $|\text{supp}(\mathbf{y}^*)| \leq r + \bar{\tau}$ can be found in time $\text{poly}(p_{\max}, \tau, \kappa, \langle \mathbf{n}, \mathbf{m}, \mathbf{p}, \mathbf{w} \rangle)$.*

4 Conclusions and Research Directions

On the side of theory, one may wonder why not apply the approach developed here to other scheduling problems, in particular those modeled as quadratic huge N -fold IP in [30], such as $R|r_j, d_j|\sum w_j C_j$. The answer is simple: we are not aware of a way to solve the separation problem in polynomial time; in fact, we believe this to be a hard problem roughly corresponding to UNARY VECTOR PACKING in variable dimension. However, the typical use of Configuration LP is not to obtain an exact optimum (which is often hard), but to obtain an approximation which is good enough. Perhaps a similar approach within our context may lead to so-called *lossy kernels* [36]? However, it is not even clear that an approximate analogue of Theorem 3 holds, because getting an LP solution whose value is close to optimal does not immediately imply getting a solution which is (geometrically) close to some optimum; cf. the discussions on ϵ -accuracy in [11, Definition 31] and [25, Introduction]. Another interesting problem highlighted here is to find a combinatorial algorithm computing the Carathéodory decomposition of the average configuration \mathbf{n}/m into machine configurations. The only approach we are aware of so far uses the equivalence of separation and optimization (thus, the ellipsoid method), which is impractical.

References

- 1 Iskander Aliev, Jesús A. De Loera, Friedrich Eisenbrand, Timm Oertel, and Robert Weismantel. The support of integer optimal solutions. *SIAM Journal on Optimization*, 28(3):2152–2157, 2018.
- 2 Matthias Bentert, René van Bevern, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Polynomial-time preprocessing for weighted problems beyond additive goal functions. *CoRR*, abs/1910.00277, 2019. [arXiv:1910.00277](https://arxiv.org/abs/1910.00277).
- 3 Hans L. Bodlaender and Michael R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995. [doi:10.1016/0167-6377\(95\)00031-9](https://doi.org/10.1016/0167-6377(95)00031-9).
- 4 Robert Brederick, Andrzej Kaczmarczyk, Dušan Knop, and Rolf Niedermeier. High-multiplicity fair allocation: Lenstra empowered by n-fold integer programming. In Anna Karlin, Nicole Immorlica, and Ramesh Johari, editors, *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019*, pages 505–523. ACM, 2019. [doi:10.1145/3328526.3329649](https://doi.org/10.1145/3328526.3329649).
- 5 Laurent Bulteau, Danny Hermelin, Dušan Knop, Anthony Labarre, and Stéphane Vialette. The clever shopper problem. *Theory of Computing Systems*, 64(1):17–34, 2020. [doi:10.1007/s00224-019-09917-z](https://doi.org/10.1007/s00224-019-09917-z).
- 6 Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Annal of Pure Applied Logic*, 84(1):119–138, 1997. [doi:10.1016/S0168-0072\(95\)00020-8](https://doi.org/10.1016/S0168-0072(95)00020-8).
- 7 Timothy Chan, Jacob W. Cooper, Martin Koutecký, Daniel Král, and Kristýna Pekárková. A row-invariant parameterized algorithm for integer programming. *CoRR*, abs/1907.06688, 2019. [arXiv:1907.06688](https://arxiv.org/abs/1907.06688).
- 8 Steven Chaplick, Fedor V. Fomin, Petr A. Golovach, Dušan Knop, and Peter Zeman. Kernelization of graph hamiltonicity: Proper h-graphs. 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 296–310. Springer, 2019. [doi:10.1007/978-3-030-24766-9_22](https://doi.org/10.1007/978-3-030-24766-9_22).
- 9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. [doi:10.1007/978-3-319-21275-3](https://doi.org/10.1007/978-3-319-21275-3).
- 10 Friedrich Eisenbrand, Christoph Hunkenschroder, and Kim-Manuel Klein. Faster algorithms for integer programs with block structure. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPICs*, pages 49:1–49:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. [doi:10.4230/LIPICs.ICALP.2018.49](https://doi.org/10.4230/LIPICs.ICALP.2018.49).
- 11 Friedrich Eisenbrand, Christoph Hunkenschroder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *CoRR*, abs/1904.01361, 2019. [arXiv:1904.01361](https://arxiv.org/abs/1904.01361).
- 12 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Operations Research Letters*, 34(5):564–568, 2006.
- 13 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. *Journal of Computer and System Sciences*, 84:1–10, 2017. [doi:10.1016/j.jcss.2016.06.004](https://doi.org/10.1016/j.jcss.2016.06.004).
- 14 Michael R. Fellows and Catherine McCartin. On the parametric complexity of schedules to minimize tardy tasks. *Theoretical Computer Science*, 298(2):317–324, 2003. [doi:10.1016/S0304-3975\(02\)00811-3](https://doi.org/10.1016/S0304-3975(02)00811-3).
- 15 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

- 16 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.
- 17 Tomáš Gavenčík, Dušan Knop, and Martin Koutecký. Integer programming in parameterized complexity: Three miniatures. In Christophe Paul and Michal Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation, IPEC 2018, August 20–24, 2018, Helsinki, Finland*, volume 115 of *LIPICs*, pages 21:1–21:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.IPEC.2018.21.
- 18 P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.
- 19 Steffen Goebels, Frank Gurski, Jochen Rethmann, and Eda Yilmaz. Change-making problems revisited: a parameterized point of view. *Journal of Combinatorial Optimization*, 34(4):1218–1236, November 2017. doi:10.1007/s10878-017-0143-z.
- 20 Michel X. Goemans and Thomas Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proc. SODA 2014*, pages 830–839, 2014.
- 21 Frank Gurski, Carolin Rehs, and Jochen Rethmann. Knapsack problems: A parameterized point of view. *Theoretical Computer Science*, 775:93–108, 2019. doi:10.1016/j.tcs.2018.12.019.
- 22 Danny Hermelin, Judith-Madeleine Kubitzka, Dvir Shabtay, Nimrod Talmon, and Gerhard J. Woeginger. Scheduling two competing agents when one agent has significantly fewer jobs. In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16–18, 2015, Patras, Greece*, volume 43 of *LIPICs*, pages 55–65. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPICs.IPEC.2015.55.
- 23 Danny Hermelin, Michael Pinedo, Dvir Shabtay, and Nimrod Talmon. On the parameterized tractability of single machine scheduling with rejection. *European Journal of Operational Research*, 273(1):67–73, 2019. doi:10.1016/j.ejor.2018.07.038.
- 24 Danny Hermelin, Dvir Shabtay, and Nimrod Talmon. On the parameterized tractability of the just-in-time flow-shop scheduling problem. *Journal of Scheduling*, 22(6):663–676, 2019. doi:10.1007/s10951-019-00617-7.
- 25 D. S. Hochbaum and J. G. Shantikumar. Convex separable optimization is not much harder than linear optimization. *J. ACM*, 37(4):843–862, 1990.
- 26 Bart M. P. Jansen and Stefan Kratsch. A structural approach to kernels for ilps: Treewidth and total unimodularity. 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 779–791. Springer, 2015. doi:10.1007/978-3-662-48350-3_65.
- 27 Klaus Jansen, Marten Maack, and Roberto Solis-Oba. Structural parameters for scheduling with assignment restrictions. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24–26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 357–368, 2017. doi:10.1007/978-3-319-57586-5_30.
- 28 Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, FOCS '82*, pages 312–320, Washington, DC, USA, 1982. IEEE Computer Society.
- 29 Dušan Knop and Martin Koutecký. Scheduling meets n -fold integer programming. *Journal of Scheduling*, 21:493–503, 2018.
- 30 Dušan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Multitype integer monoid optimization and applications. *CoRR*, abs/1909.07326, 2019. arXiv:1909.07326.
- 31 Dušan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n -fold integer programming and applications. *Mathematical Programming*, November 2019. doi:10.1007/s10107-019-01402-2.

- 32 Dušan Knop, Martin Koutecký, and Matthias Mnich. Voting and bribing in single-exponential time. *ACM Transactions on Economics and Computation*, 8(3), June 2020. doi:10.1145/3396855.
- 33 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPICs*, pages 85:1–85:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.85.
- 34 Stefan Kratsch. On polynomial kernels for integer linear programs: Covering, packing and feasibility. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 647–658. Springer, 2013. doi:10.1007/978-3-642-40450-4_55.
- 35 Stefan Kratsch. On polynomial kernels for sparse integer linear programs. *Journal of Computer and System Sciences*, 82(5):758–766, 2016. doi:10.1016/j.jcss.2015.12.002.
- 36 Daniel Lokshtanov, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 224–237, 2017.
- 37 Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, 2018.
- 38 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Math. Program.*, 154(1-2):533–562, 2015.
- 39 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. doi:10.1093/ACPROF:OSO/9780198566076.001.0001.
- 40 Guntram Scheithauer and Johannes Terno. The modified integer round-up property of the one-dimensional cutting stock problem. *European Journal of Operational Research*, 84(3):562–571, 1995.
- 41 René van Bevern, Till Fluschnik, and Oxana Yu. Tsidulko. On $(1+\epsilon)$ -approximate data reduction for the rural postman problem. In Michael Khachay, Yury Kochetov, and Panos M. Pardalos, editors, *Mathematical Optimization Theory and Operations Research - 18th International Conference, MOTOR 2019, Ekaterinburg, Russia, July 8-12, 2019, Proceedings*, volume 11548 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2019. doi:10.1007/978-3-030-22629-9_20.
- 42 René van Bevern, Till Fluschnik, and Oxana Yu. Tsidulko. Parameterized algorithms and data reduction for the short secluded s-t-path problem. *Networks*, 75(1):34–63, 2020. doi:10.1002/net.21904.
- 43 René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 18(5):449–469, 2015. doi:10.1007/s10951-014-0398-5.

Abstract Morphing Using the Hausdorff Distance and Voronoi Diagrams

Lex de Kogel

Utrecht University, The Netherlands

Marc van Kreveld

Utrecht University, The Netherlands

Jordi L. Vermeulen

Utrecht University, The Netherlands

Abstract

This paper introduces two new abstract morphs for two 2-dimensional shapes. The intermediate shapes gradually reduce the Hausdorff distance to the goal shape and increase the Hausdorff distance to the initial shape. The morphs are conceptually simple and apply to shapes with multiple components and/or holes. We prove some basic properties relating to continuity, containment, and area. Then we give an experimental analysis that includes the two new morphs and a recently introduced abstract morph that is also based on the Hausdorff distance [23]. We show results on the area and perimeter development throughout the morph, and also the number of components and holes. A visual comparison shows that one of the new morphs appears most attractive.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Morphing, Hausdorff distance, Voronoi diagrams

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.74

Related Version *Full Version:* <https://arxiv.org/abs/2206.15339>

Supplementary Material *InteractiveResource:* <https://hausdorff-morphing.github.io/>

Funding Research was funded by NWO TOP grant no. 612.001.651.

1 Introduction

Morphing, also referred to as shape interpolation, is the changing of a given shape into a target shape over time. Applications include animation and medical imaging. Animation is often motivated by the film industry, where morphing can be used to create cartoons or visual effects. In medical imaging, the objective is a 3D reconstruction from cross-sections, such as those from MRI or CT scans. Reconstruction between two 2D slices is essentially 2D interpolation between shapes, which is a form of morphing. We regard morphing itself as the change of one shape into another shape by a parameter, or, more precisely, a function from the interval $[0, 1]$ to shapes in a space, such that the image at 0 is the one input shape and the image at 1 is the other input shape. It is often convenient to see the morphing parameter as time. In the rest of this paper, we will refer to the shape of the morph at any particular time value as an *intermediate shape*. See Figure 1 for an example of two halfway shapes between polygons resembling a butterfly and a spider.

The quality of a morph depends on the application. For medical imaging, the implied 3D reconstruction must be anatomically plausible. For morphing between two drawings of a cartoon character, the shapes in between must keep the dimensions of the limbs, for example. Furthermore, semantically meaningful features (nose, chin) should morph from their position in the one shape to their position in the other shape.



© Lex de Kogel, Marc van Kreveld, and Jordi L. Vermeulen;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 74; pp. 74:1–74:16

Leibniz International Proceedings in Informatics



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



■ **Figure 1** The intermediate shapes of two different morphing methods at time value $1/2$ when morphing between the input shapes on the left. The middle shows the dilation morph (introduced in [23]), the right shows the Voronoi morph (introduced in this paper).

In this paper we concentrate on abstract morphing of shapes. A morphing task is abstract if there is no (semantic) reason to transform certain parts of a starting shape into certain parts of a goal shape. In a recent paper, van Kreveld et al. [23] presented a new type of abstract morphing based on the Hausdorff distance. It takes any two compact planar shapes A and B as input, and produces a morphed shape that interpolates smoothly between them. For a time value $\alpha \in [0, 1]$, this morph is equal to A at $\alpha = 0$ and to B at $\alpha = 1$. For any value of α it has Hausdorff distance α to A and Hausdorff distance $1 - \alpha$ to B , if the initial Hausdorff distance is 1 (the input can be scaled to make this true without changing intermediate shapes). Morphs with this property are called *Hausdorff morphs* [23]. The Hausdorff morph introduced by van Kreveld et al. is based on Minkowski sums with a disk, and hence we refer to this specific one as the *dilation morph*.

While the dilation morph has nice theoretical properties, in practice it will often grow intermediate shapes from A until $\alpha = 1/2$, at which point the greatly dilated shape will shrink back towards B . For α close to $1/2$, the morphed shape typically resembles neither of the input shapes unless they already looked alike. We can see this in Figure 1.

In this paper we present a new Hausdorff morph called *Voronoi morph* that gives a more visually convincing morph, while maintaining many of the properties of the previous work. Our morph uses Voronoi diagrams to partition each input shape into regions with the same closest point on the other shape, and then scales and moves each such region to that closest point based on the value of α . We show that the Voronoi morph is also a Hausdorff morph. It interpolates smoothly between A and B , but does not have the same problem of significantly increasing the area during the morph. We also present a variant called *mixed morph* that reduces the problem of unnecessarily increasing the perimeter of the interpolated shape. It uses dilation and erosion to overcome some shortcomings of the Voronoi morph.

Related work. The Hausdorff distance is a widely used distance metric that can be used for any two subsets of a space. It is a bottleneck measure: only a maximum distance determines the Hausdorff distance. Efficient algorithms to compute the Hausdorff distance between two simple polygons or their higher-dimensional equivalents exist [3, 4, 7]. The Hausdorff distance is used in computer vision [19] and computer graphics [6, 17] for template matching, and the computation of error between a model and its simplification.

Several algorithmic approaches to morphing have been described. Many of these are motivated by shape interpolation between slices (e.g., [2, 9, 10, 12], an overview can be found in [11]). Other papers discuss morphing explicitly and not as an interpolation problem. Many of these results use compatible triangulations [20, 24], in particular those that avoid self-intersections. It is beyond the scope of this paper to give a complete overview of morphing methods. For (not so recent) surveys of shape matching, interpolation, and correspondence, see [5, 22]. Our paper builds upon the morphing approach given in [23], which introduced Hausdorff morphs as a new technique for abstract morphing, and the dilation morph as a specific example of a Hausdorff morph.

Another shape similarity measure than the Hausdorff distance, the *Fréchet* distance, can also be used to define a morph. In particular, *locally correct Fréchet matchings* [14] immediately imply a smooth transition of one shape outline into another, because they match all pairs of points on the two curves. Similar approaches were given in [25, 26]. During the transition, however, the outline may be self-intersecting. This problem was addressed in [15, 16]. A more important shortcoming of morphing using the Fréchet distance is that it is unclear how to morph between shapes with different numbers of components and holes.

Much of the commercial software for morphing applies to images, with or without additional human control. Other software is meant as toolkits for designers to design their own morphs, most notably Adobe After Effects.

Our results. We introduce two new abstract morphs based on the Hausdorff distance. They are – just like the dilation morph – conceptually simple and easy to implement if one has code for Minkowski sum and difference with a disk, Voronoi diagrams, and polygon intersection and union. We examine basic properties of the two new morphs and compare how they relate to the dilation morph. In particular, we show that the Voronoi morph is a Hausdorff morph and that it is 1-Lipschitz continuous. We also show that for any morphing parameter (time), the Voronoi-morph intermediate shape is a subset of the mixed-morph intermediate shape, which in turn is a subset of the dilation-morph intermediate shape.

We then proceed with an extensive experimental analysis where we compare four basic quantities: area, perimeter, number of components, and number of holes. We show how these quantities develop throughout the three morphs. We also present visual results. As data we use simple drawings of animals, country outlines, and text (letters and whole words).

2 Preliminaries

Given two sets A and B , we can define the directed Hausdorff distance from A to B as

$$d_{\vec{H}}(A, B) := \sup_{a \in A} \inf_{b \in B} d(a, b),$$

where d denotes the Euclidean distance. The *undirected Hausdorff distance* between A and B is then defined as the maximum of both directed distances:

$$d_H(A, B) := \max(d_{\vec{H}}(A, B), d_{\vec{H}}(B, A)).$$

When A and B are closed sets, we can alternatively define the Hausdorff distance using Minkowski sums. Recall that the Minkowski sum $A \oplus B$ is defined as $\{a + b \mid a \in A, b \in B\}$; the directed Hausdorff distance between A and B is then the smallest value r for which $A \subseteq B \oplus D_r$, where D_r is a disk of radius r .

Van Kreveld et al. [23] then define a function that interpolates between two shapes in a Hausdorff sense: For any time parameter $\alpha \in [0, 1]$, they define the dilation morph

$$S_\alpha(A, B) := (A \oplus D_\alpha) \cap (B \oplus D_{1-\alpha}),$$

and prove that this shape has Hausdorff distance α to A and $1 - \alpha$ to B , and that it is the maximal shape with this property. Additionally, they show that this morph is 1-Lipschitz continuous: for two time parameters α and β , $d_H(S_\alpha(A, B), S_\beta(A, B)) \leq |\beta - \alpha|$. Note that we will omit the arguments A, B when they are clear from context.

Structurally, it turns out that the intermediate shapes may have quadratic complexity, even when the input is two simple polygons with n vertices each. For instance, if the input consists of a horizontal comb and an overlapping vertical comb, each with $n/4$ prongs, S_α will consist of $\Omega(n^2)$ components for any $\alpha \in (0, 1)$. In fact, this is not limited to the dilation morph: *any* intermediate shape with Hausdorff distance α to A and $1 - \alpha$ to B will have $\Omega(n^2)$ components [23], so every Hausdorff morph has this feature.

Note that both S_α and the morphing methods described below change when we translate one of the input shapes. That is, if we write $t + B$ to be the translation of B over a vector t , it is not true that $S_\alpha(A, t + B) = \alpha * t + S_\alpha(A, B)$, as one might expect. This is because the positions of the input shapes are important both for the Hausdorff distance and for the shape of S_α . However, we can simply calculate $S_\alpha(A, B)$ with A and B aligned in some way, and then generate $S_\alpha(A, t + B)$ by explicitly translating $S_\alpha(A, B)$ by $\alpha * t$. Sensible alignment methods include aligning the centroids, maximising the overlap of the shapes, and minimising the Hausdorff distance.

3 Voronoi morph

As demonstrated in Figure 1, one of the problems with the dilation morph is that the intermediate shapes tend to lose any resemblance to the input during the morphing process. The main reason for this is that the dilated shapes we are intersecting contain many points that do not influence the Hausdorff distance in any way, because they are not on the shortest path from a point on one shape to the closest point on the other. In other words, much of S_α can be removed without changing the Hausdorff distance to and from the input. That said, there is no obvious “correct” way to determine which parts should be removed to obtain the greatest resemblance to the input.

We propose a morph in which we only take the points of S_α that are on the shortest path between points in one input shape and the closest point on the other. Specifically, we only take the points where the ratio of distances to the one shape and the closest point on the other is $\alpha : 1 - \alpha$. More formally, we define our new morph T_α as follows:

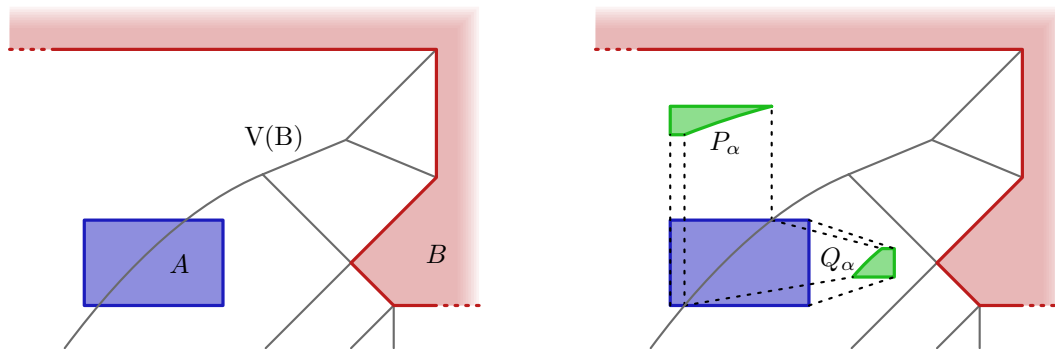
$$T_\alpha(A, B) := \{a + \alpha(c(a, B) - a) \mid a \in A\} \cup \{b + (1 - \alpha)(c(b, A) - b) \mid b \in B\},$$

where $c(a, B)$ denotes the point on B closest to a . In other words, we move each point in A closer to the closest point in B by a fraction α of that distance, and each point in B closer to the closest point in A by a fraction $1 - \alpha$, and take the union of those two shapes. If a point is equidistant to multiple points in the other shape, we include all options. We can prove that this morph has the desired Hausdorff distances to the input.

► **Theorem 1.** *Let A and B be two compact sets in the plane with $d_H(A, B) = 1$. Then for any $0 \leq \alpha \leq 1$, we have $d_H(A, T_\alpha) = \alpha$ and $d_H(B, T_\alpha) = 1 - \alpha$.*

Proof. We first show that $d_H(A, T_\alpha) \leq \alpha$, and then show strict equality. The case for $d_H(B, T_\alpha)$ is analogous and therefore omitted.

By construction, any point $a \in A$ has a point at distance $\leq \alpha$ in T_α , showing that $d_H(A, T_\alpha) \leq \alpha$. Similarly, by construction, for each point $b \in B$ there is a point $t_b \in T_\alpha$ such that $t_b = (1 - \alpha)(c(b, A) - b)$. As $d(b, c(b, A)) \leq 1$, it must be the case that t_b has distance



■ **Figure 2** On the left, A is partitioned by the Voronoi diagram $V(B)$ of B . On the right, each partitioned part of A , shown in green, is scaled towards the closest point on B by a factor α .

at most α to $c(b, A)$. It follows that all points in T_α have distance at most α to a point in A , thereby showing that $d_{\bar{H}}(T_\alpha, A) \leq \alpha$. As we have both $d_{\bar{H}}(A, T_\alpha) \leq \alpha$ and $d_{\bar{H}}(T_\alpha, A) \leq \alpha$, it follows that $d_H(A, T_\alpha) \leq \alpha$.

To show strict equality, assume the Hausdorff distance is realised by some point $\hat{a} \in A$ with closest point $\hat{b} \in B$, i.e., $d(\hat{a}, \hat{b}) = 1$. By construction, there is a point $\hat{t} \in T_\alpha$ at distance α from \hat{a} and at distance $1 - \alpha$ from \hat{b} . As \hat{t} is the closest point to \hat{a} in T_α , we have $d_H(A, T_\alpha) = \alpha$, and as \hat{b} is the closest point to \hat{t} in B , we have $d_H(B, T_\alpha) = 1 - \alpha$. If the Hausdorff distance is realised by a point on B , we use a symmetric argument. ◀

We can additionally show that the Voronoi morph, like the dilation morph, is 1-Lipschitz continuous in a Hausdorff sense:

▶ **Lemma 2.** *Let $\alpha, \beta \in [0, 1]$. Then $d_H(T_\alpha, T_\beta) \leq |\beta - \alpha|$.*

Proof. Let t_α be any point on T_α . Assume without loss of generality that there is some $a \in A$ such that $t_\alpha = a + \alpha(c(a, B) - a)$ (the case for t_α being included due to a point in B is analogous). Now consider the point $t_\beta = a + \beta(c(a, B) - a)$: t_α and t_β are on the same straight line segment between a and $c(a, B)$, and have distance $|\beta - \alpha| \cdot |c(a, B) - a|$ to each other. As $d_H(A, B) = 1$, we know that $|c(a, B) - a| \leq 1$, and therefore that $|t_\beta - t_\alpha| \leq |\beta - \alpha|$. This holds for any $t_\alpha \in T_\alpha$, and the argument is symmetric for T_β . ◀

Note that this type of continuity implies that components of T_α can only form or disappear by merging with or splitting from another component.

In addition to the Hausdorff distance-related properties, it is also interesting to study the general geometric and topological properties of T_α . We first show that the number of components $\#C(T_\alpha)$ of T_α does not change during the morph, except possibly at $\alpha = 0$ and $\alpha = 1$. We prove this for the case of polygonal input; the proof can likely be generalised, but the formalisation is somewhat tedious and not particularly interesting.

Let $V(A)$ be the Voronoi diagram of the vertices, open edges and the interior components of A . We now define $\text{Par}(A, B)$ to be the input shape A partitioned into regions by $V(B)$. Note that $\text{Par}(A, B)$ is a set of regions of A that each have the closest point of B on the same vertex, edge or face of B . For some region $P \in \text{Par}(A, B)$, let P_α be the region obtained by scaling P towards the site of the Voronoi cell of B it is in by a factor α . If this site is a vertex, we simply scale P uniformly towards it; if it is an edge, we scale it perpendicular to the supporting line of that edge; and if it is a face, it does not scale or move at all; see Figure 2 for an illustration. Now let $\text{Par}_\alpha(A, B) := \{P_\alpha \mid P \in \text{Par}(A, B)\}$. Note that T_α is the union of all elements of $\text{Par}_\alpha(A, B)$ and $\text{Par}_{1-\alpha}(B, A)$.

► **Lemma 3.** *Let $0 < \alpha < \beta < 1$. Then $\#C(T_\alpha) = \#C(T_\beta)$.*

Proof. Assume that $\#C(T_\alpha) \neq \#C(T_\beta)$. We can assume without loss of generality that $\#C(T_\alpha) > \#C(T_\beta)$, as in the other case we can take $T_\alpha(B, A)$ instead of $T_\alpha(A, B)$ and get the same morph, but parametrised in reverse. We can also assume that for fixed α, β is the smallest value such that $\#C(T_\alpha) > \#C(T_\beta)$. In this case, there are two regions P and Q of $\text{Par}(A, B)$ or $\text{Par}(B, A)$ that are disjoint and in different components of T_α , but intersect and are in the same component of T_β . This is because, as a consequence of Lemma 2, components cannot appear or disappear. In the following we assume $P, Q \in \text{Par}(A, B)$; the arguments for when one or both are in $\text{Par}(B, A)$ are identical.

As $P_\beta \cap Q_\beta \neq \emptyset$, there must be some point p in both P_β and Q_β . As both P_β and Q_β are formed by regions moving towards the closest point on the other shape, this point is then on the intersection of two shortest paths between A and B . Let a_1, b_1, a_2 and b_2 be the endpoints of these paths intersecting in p . One of the two segments $\overline{pb_1}, \overline{pb_2}$ will be the shortest; assume without loss of generality that it is $\overline{pb_1}$. In this case the path a_2pb_1 is shorter than a_2pb_2 , and by the triangle inequality b_1 must be closer to a_2 than b_2 .

This contradicts the assumption that b_2 was the closest point to a_2 . We conclude that such shortest paths can never intersect, and therefore $P_\alpha \cap Q_\alpha = \emptyset$ for any $\alpha \in (0, 1)$. As such, components can never merge or split for $\alpha \in (0, 1)$, and as they also cannot appear or disappear by Lemma 2, the statement in the lemma follows. ◀

Note that the number of components can change at $\alpha = 0$ or $\alpha = 1$, as in these limit cases elements of $\text{Par}_\alpha(A, B)$ and $\text{Par}_\alpha(B, A)$ turn into points or line segments. Using the strategy from this proof, it also follows that $\text{Par}_\alpha(A, B)$ and $\text{Par}_{1-\alpha}(B, A)$ are interior-disjoint. An interesting corollary of this observation is that the area $|T_\alpha|$ of T_α is bounded from below by $(1 - \alpha)^2 |A| + \alpha^2 |B|$, which is attained when both shapes are disjoint and all parts are moving to a finite number of points (vertices) on the other shape.

3.1 A variant morph

One problem with the Voronoi morph is that it can introduce many slits into the boundary, thereby greatly increasing the perimeter of the shape. This is because parts of the input that have different closest points on the other shape will tend to move away from each other. We present a variant of the Voronoi morph that tries to reduce these problems. As it uses both the Voronoi morph and the dilation morph, we call this variant the *mixed morph*. The mixed morph $M_{\alpha, \varphi}$ is defined as follows:

$$M_{\alpha, \varphi}(A, B) := ((T_\alpha(A, B) \oplus D_\varphi) \ominus D_\varphi) \cap S_\alpha,$$

where \ominus is the Minkowski difference, defined as $A \ominus B := (A^c \oplus B)^c$, where A^c is the complement of A . Taking a Minkowski sum with a disk is also known as *dilation*, and the Minkowski difference with a disk is known as *erosion*. Performing first a dilation and then an erosion with disks of the same radius is known as *closing*, and can be used to close small gaps and holes in a shape without modifying the rest too much. The closing operator is widely used and studied in the field of image analysis [21].

The resulting morph may no longer be a Hausdorff morph: we may have increased the Hausdorff distance by closing certain gaps or holes. We therefore intersect the closed version of T_α with the dilation morph S_α , so that gaps that are necessary to obtain the appropriate Hausdorff distance are maintained. This results in the mixed morph $M_{\alpha, \varphi}$.

The mixed morph has a new parameter, φ , being the radius of the disk used in the closing. Note that $M_{\alpha,0} = T_\alpha$. We can show that $M_{\alpha,\varphi}$ contains all shapes obtained with the same α but smaller value of φ :

► **Lemma 4.** *Let $\varphi, \psi \in \mathbb{R}^+$ and $\varphi \leq \psi$. Then $M_{\alpha,\varphi} \subseteq M_{\alpha,\psi}$.*

Proof. Let us assume that $M_{\alpha,\varphi} \supset M_{\alpha,\psi}$ instead. Then there is some point p such that $p \in M_{\alpha,\varphi}$, but $p \notin M_{\alpha,\psi}$. There are two reasons why p might not be in $M_{\alpha,\psi}$: either $p \notin T_\alpha \oplus D_\psi$, or $p \in T_\alpha \oplus D_\psi$ but $p \notin (T_\alpha \oplus D_\psi) \ominus D_\psi$.

It can clearly not be the case that $p \in M_{\alpha,\varphi}$ but $p \notin T_\alpha \oplus D_\psi$: $M_{\alpha,\varphi}$ is a subset of $T_\alpha \oplus D_\varphi$, and as $\varphi \leq \psi$, we have that $T_\alpha \oplus D_\varphi \subseteq T_\alpha \oplus D_\psi$.

It must then be the case that $p \in T_\alpha \oplus D_\psi$ but $p \notin (T_\alpha \oplus D_\psi) \ominus D_\psi$. In this case, the distance between p and the boundary ∂T_α^\oplus of $T_\alpha \oplus D_\psi$ must be less than ψ . Let $q \in \partial T_\alpha^\oplus$ be the point on the boundary closest to p . As $p \in T_\alpha \oplus D_\varphi$ and $T_\alpha \oplus D_\varphi \subseteq T_\alpha \oplus D_\psi$, the segment \overline{pq} must intersect the boundary of $T_\alpha \oplus D_\varphi$ in some point q' . We must have that $d(p, q') \geq \varphi$, or p would not be in $M_{\alpha,\varphi}$, and we must have $d(q, q') \geq \psi - \varphi$, as $T_\alpha \oplus D_\psi = (T_\alpha \oplus D_\varphi) \oplus D_{\psi-\varphi}$. But then, by the triangle inequality, $d(p, q) \leq d(p, q') + d(q, q') \geq \psi$, which is a contradiction. Hence, $p \in M_{\alpha,\psi}$. As this holds for all $p \in M_{\alpha,\varphi}$, the statement in the lemma follows. ◀

Note that this means we now have the following hierarchical containment of morphs: $T_\alpha \subseteq M_{\alpha,\varphi} \subseteq M_{\alpha,\psi} \subseteq S_\alpha$, for $\varphi \leq \psi$. As T_α is a Hausdorff morph, and S_α is the maximal Hausdorff morph, this shows that $M_{\alpha,\varphi}$ is a Hausdorff morph as well. However, $M_{\alpha,\varphi}$ is not 1-Lipschitz continuous: components may suddenly merge when their distance falls below 2φ .

3.2 Algorithm

To give an algorithm for computing T_α , we assume A and B are (sets of) polygons, possibly with holes. As T_α is based on moving all points on the one shape to the closest point on the other shape, we can compute the Voronoi diagram of each input shape, and then use these to partition the other shapes. This gives us a partitioning of A into pieces that overlap B , or have the same closest point or edge on B , and vice versa. Pieces of A completely inside B are unchanged, pieces with a vertex as closest element are scaled uniformly towards that vertex by a factor α , and pieces with an edge as closest element are scaled perpendicular to the supporting line of that edge by a factor α . For pieces of B we do the same, except that we scale them with a factor $1 - \alpha$. Figure 2 shows an example of how a shape A is partitioned by the Voronoi diagram $V(B)$ of B , and each piece is scaled towards the closest point on B .

Given this algorithm, we can also straightforwardly compute $M_{\alpha,\varphi}$ by computing T_α and S_α , dilating and eroding T_α by a distance φ , and then intersecting the result with S_α .

Our computations rely solely on Voronoi diagrams, Minkowski sums and differences with disks, intersections and unions of polygons, all of which can be found in standard books or surveys [1, 8, 18] and an intermediate shape can be calculated in $O(n^2 \log n)$ time.

4 Experiments

We compare the dilation, Voronoi and mixed morphs experimentally on three data sets. The first data set is a collection of outlines of animals taken from [13]. The second is a selection of the outlines of European countries obtained from the Thematic Mapping World Borders data set;¹ we use the outlines of Austria, Belgium, Croatia, Czechia, France, Germany, Greece, Ireland, Italy, the Netherlands, Poland, Spain and Sweden. For these two sets we compute

¹ http://www.thematicmapping.org/downloads/world_borders.php

the morphs for all pairs of animals and all pairs of countries in the sets. None of the three morphs is translation-invariant or scale-invariant, so it matters where we place the shapes with respect to each other and what sizes they initially have. We choose to scale the shapes to have the same area and translate them to have a common centroid.

The third data set is a small collection of words and letters manually traced as polygons. We use three pairs of words (wish/luck, kick/stuff, try/it), and the letters f, i and u in a serif and a sans serif font. Observe that our morphs could in theory be used to define an infinite family of fonts by interpolating between the glyphs of each element. For these experiments we do not scale the shapes but use the font size, and we align them manually.

For each experiment, we measure the area, perimeter, number of components and number of holes of the morph for α values starting at zero and increasing in steps of $1/8$. The parameter φ of the mixed morph was universally set to 0.02 based on preliminary experimentation.

It is not necessarily insightful to compare areas and especially perimeters between experiments. To make the results more comparable, we make the assumption that an ideal morph linearly interpolates the area and perimeter between those of the input shapes. For each experiment, we can then give the ratio between the measured area and perimeter and these “ideal” values. For the number of components and holes this is less meaningful, as these are discrete values, so we simply record the numbers directly.

Each morphing method was implemented in C++, using Boost² to calculate intersections and unions of polygons, Voronoi diagrams, and Minkowski sums. Although efficiency is not the focus of this paper, running all our experiments only took a few minutes in total.

5 Results

A summary of our measurements of area and perimeter can be seen in Tables 1 and 2. A summary of the number of components and holes for only the animals data set can be seen in Table 3; we exclude the other data sets because the inputs have different numbers of components. Topological measurements for the animals data set can be viewed in Table 4; the measurements for the other experiments can be viewed in the full version. We note that the Voronoi and mixed morphs sometimes have spurious holes caused by numerical precision issues (e.g., the Voronoi morph should not have an intermediate shape with five holes in our experiment with the letter i). Animations of the different morphs for each experiment can be viewed online.³

In Figure 3 we can see that the average area of the dilation morph quickly grows as α increases, until reaching its peak at $\alpha = 1/2$, to about three times the desired size. For the perimeter we see the opposite trend, with the dilation morph typically having a lower perimeter than desired. This is a consequence of the dilation erasing details in the boundary of the input shapes. We can see in Figure 4 that this happens more quickly in the experiments with country shapes. This is expected, as most of the country shapes have more sharp coastline features and islands that quickly disappear, whereas the animal shapes are generally smoother and only have one component.

Our Voronoi morph on average has an area that is much closer to the desired value, and with much lower variance than the dilation morph. However, we see that on average the perimeter is much higher than the desired value. This is because points on opposite sides of a Voronoi edge move in different directions, causing new boundaries to appear in the interior

² <https://www.boost.org>

³ <https://hausdorff-morphing.github.io>

■ **Table 1** The distributions of areas for each morphing method over all experiments for all nine tested values of α , separated by experiment category.

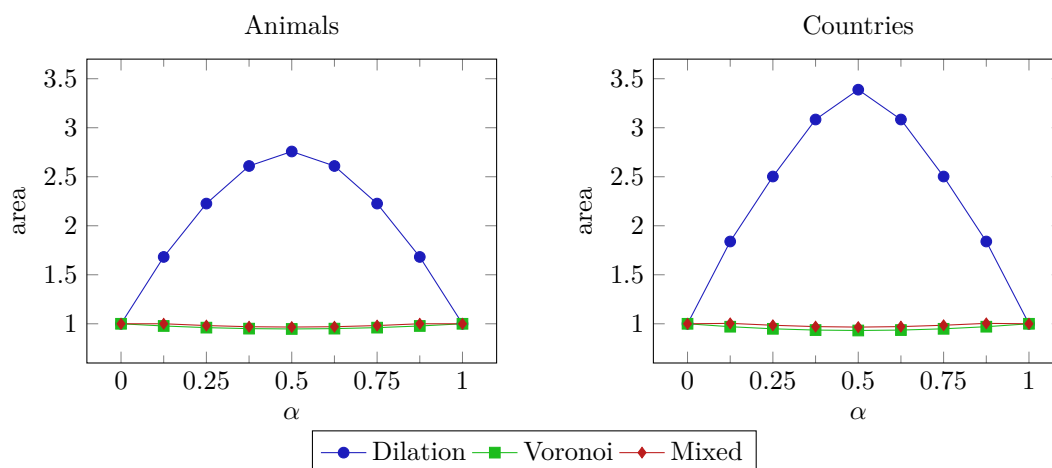
Category	Dilation		Voronoi		Mixed	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Animals	1.977	0.763	0.969	0.024	0.986	0.019
Countries	2.249	1.498	0.960	0.039	0.987	0.039
Text	2.118	1.046	0.980	0.035	0.989	0.028

■ **Table 2** The distributions of perimeters for each morphing method over all experiments for all nine tested values of α , separated by experiment category.

Category	Dilation		Voronoi		Mixed	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Animals	0.857	0.137	1.725	0.432	1.183	0.155
Countries	0.876	0.237	1.610	0.471	1.129	0.184
Text	0.955	0.142	1.401	0.418	1.155	0.192

■ **Table 3** The distributions of the number of components and holes for each morphing method for all tested values of α except 0 and 1. This only includes the animals data set, as these shapes have only one component and no holes.

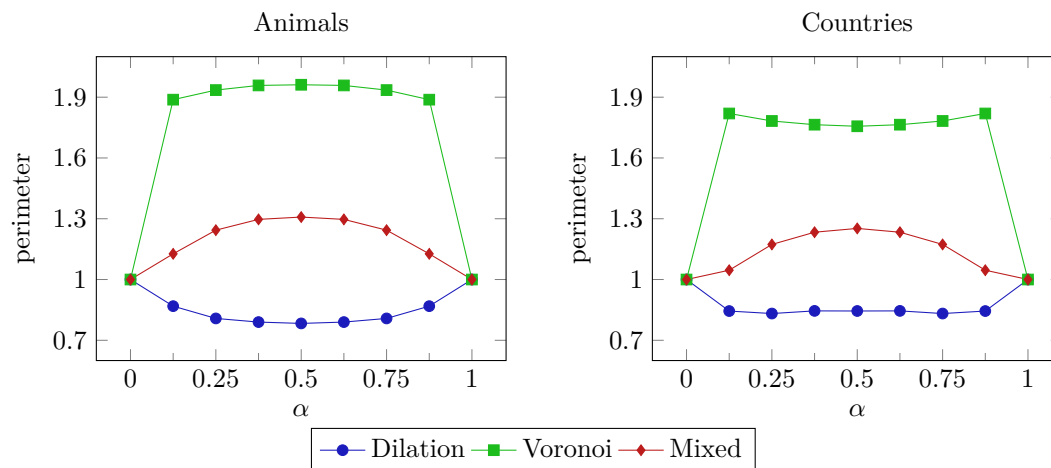
Category	Dilation		Voronoi		Mixed	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Components	1.004	0.063	18.556	8.089	5.317	3.213
Holes	0.218	0.602	2.544	2.699	0.218	0.532



■ **Figure 3** The average area over all experiments as a function of α , for both the animals and countries data sets.

■ **Table 4** The minimum and maximum number of components and the maximum number of holes for the experiments with animal shapes, separated by morph type.

Experiment	Dilation			Voronoi			Mixed		
	min	max	holes	min	max	holes	min	max	holes
bird → butterfly	1	1	0	1	11	6	1	4	1
bird → cat	1	1	1	1	12	6	1	5	0
bird → dog	1	1	1	1	23	4	1	7	1
bird → horse	1	2	1	1	27	6	1	8	2
bird → ostrich	1	1	0	1	26	5	1	12	0
bird → shark	1	1	0	1	14	4	1	6	0
bird → spider	1	1	0	1	30	8	1	9	1
bird → turtle	1	1	0	1	21	3	1	9	1
butterfly → cat	1	1	1	1	6	2	1	3	1
butterfly → dog	1	1	1	1	9	4	1	4	1
butterfly → horse	1	1	1	1	28	3	1	9	2
butterfly → ostrich	1	1	1	1	11	7	1	4	2
butterfly → shark	1	1	0	1	8	4	1	3	1
butterfly → spider	1	1	1	1	30	9	1	10	2
butterfly → turtle	1	1	0	1	13	6	1	3	2
cat → dog	1	1	1	1	17	1	1	3	1
cat → horse	1	1	1	1	17	2	1	5	1
cat → ostrich	1	1	0	1	13	4	1	4	0
cat → shark	1	1	0	1	9	0	1	5	0
cat → spider	1	1	2	1	29	3	1	7	3
cat → turtle	1	1	0	1	14	1	1	7	0
dog → horse	1	1	2	1	31	4	1	9	1
dog → ostrich	1	1	1	1	22	2	1	7	1
dog → shark	1	1	1	1	17	3	1	6	1
dog → spider	1	1	3	1	32	2	1	14	2
dog → turtle	1	1	1	1	16	1	1	6	0
horse → ostrich	1	1	2	1	27	7	1	15	1
horse → shark	1	1	1	1	22	5	1	7	1
horse → spider	1	1	4	1	38	3	1	9	1
horse → turtle	1	1	1	1	22	4	1	12	0
ostrich → shark	1	1	0	1	15	8	1	5	0
ostrich → spider	1	1	4	1	38	9	1	17	0
ostrich → turtle	1	1	0	1	21	12	1	4	0
shark → spider	1	1	0	1	23	2	1	5	2
shark → turtle	1	1	0	1	11	3	1	3	0
spider → turtle	1	1	4	1	25	14	1	8	3



■ **Figure 4** The average perimeter over all experiments as a function of α , for both the animals and countries data sets.

of a shape as soon as $\alpha > 0$. We can see this happen in the middle column of Figure 5, and this is reflected in Figure 4, where we see the perimeter sharply increase and then stay mostly the same, before sharply dropping back down.

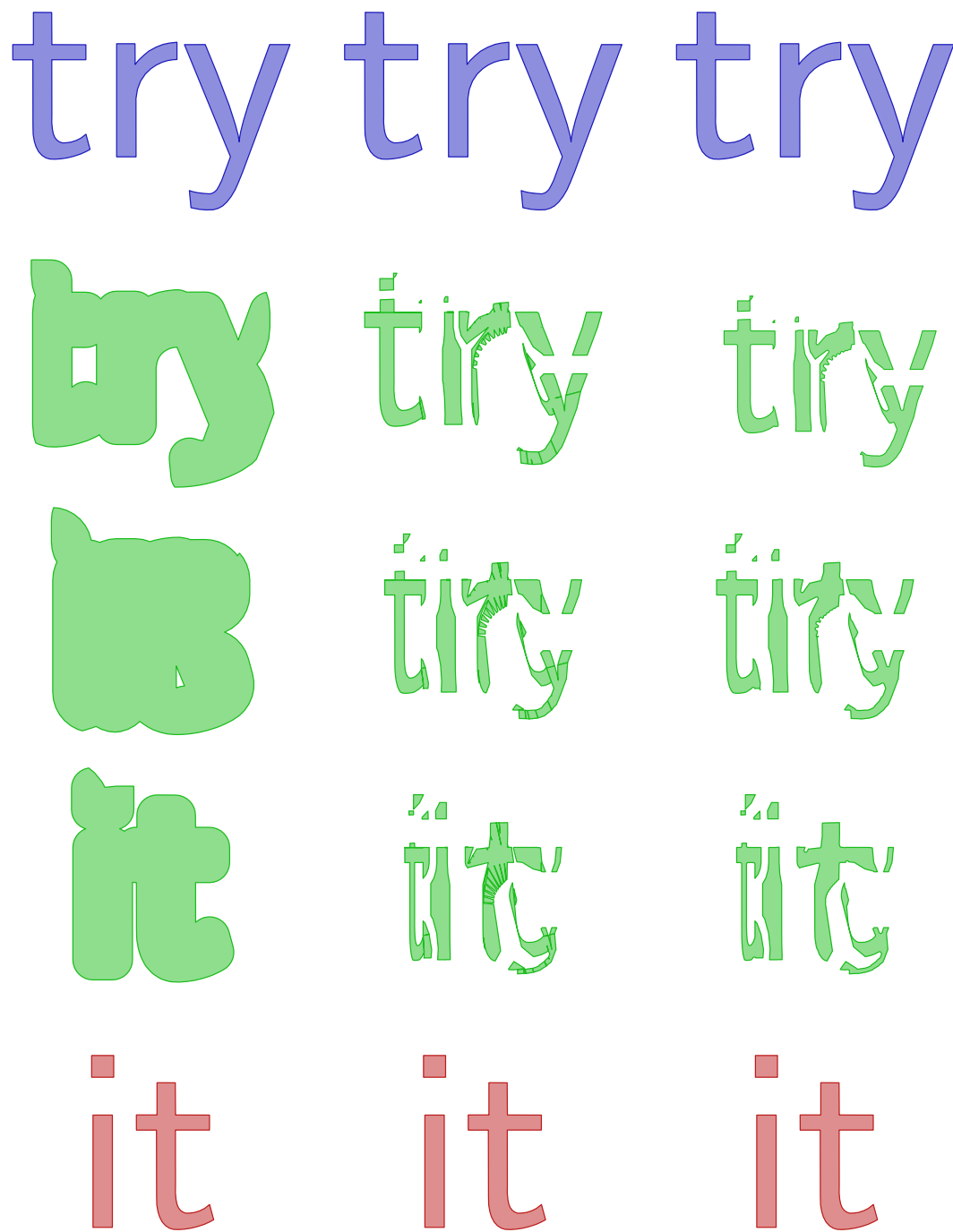
Our mixed morph achieves its purpose of reducing the perimeter of the Voronoi morph: the measured perimeters are close to the desired values, while the measured areas stay comparable to those of the Voronoi morph. In Figure 4, we see that the perimeter typically still increases during the morphing process, but does not jump up sharply as soon as $\alpha > 0$. This is because the small value of φ lets us close only the narrow gaps that appear around the edges of the Voronoi diagram, but not the gaps that develop as pieces of the shapes move apart significantly. We can see this when comparing the middle and right columns of Figure 5: fewer gaps are closed at $\alpha = 1/2$ than at the other time values.

In addition to area and perimeter, we also tracked the number of components and holes for each morph type. We observe that for the dilation morph, there is an intermediate shape with only one component in all but one of our experiments, showing that this morph tends to turn everything into a blob during the morphing process. On the other hand, the Voronoi morph tends to have an intermediate shape with a number of components much larger than either of the input shapes. The mixed morph exhibits neither of these behaviours. This is illustrated in Figure 5.

Inspecting the morphs visually (see both the full version online for more figures, as well as <https://hausdorff-morphing.github.io> for all experiments), our mixed morph looks quite reasonable, especially when the area of symmetric difference between the input shapes is small. In many cases, the intermediate shape at $\alpha = 1/2$ is a recognisable mix of the two input shapes. This is not the case for the dilation morph, where the Hausdorff distance needs to be very small compared to the size of the input shapes for it to look good. For instance, when one shape has some small islands far away, the dilation morph will grow to have a very large area, whereas with the Voronoi and mixed morphs, the islands just slowly move towards the closest point on the other shape; see the full version online. However, both the Voronoi and mixed morph can still look bad when the area of symmetric difference is large. It may therefore be best to align the input shapes such that the area of symmetric difference is minimised, rather than simply aligning the centroids.



■ **Figure 5** Intermediate shapes for $\alpha \in \{0, 1/4, 1/2, 3/4, 1\}$ when morphing between the outlines of Germany and Italy. The columns show the dilation morph, Voronoi morph and mixed morph from left to right.



■ **Figure 6** Intermediate shapes for $\alpha \in \{0, 1/4, 1/2, 3/4, 1\}$ when morphing between the outlines of the words try and it. The columns show the dilation morph, Voronoi morph and mixed morph from left to right. Note that some artefacts in the Voronoi and mixed morphs, such as in the curved part of the letter r, are caused by having polygonal input instead of smooth curves.

The morphs generally look less convincing on our experiments with text, as the shapes can be very different. For single letters the morphs can look convincing, but when morphing between words, especially of different numbers of letters, the intermediate shape at $\alpha = 1/2$ does not necessarily resemble both input shapes (see Figure 6). However, the intermediate shapes at $\alpha = 1/4$ and $\alpha = 3/4$ still do clearly resemble input shapes A and B , respectively, for the Voronoi and mixed morphs, but less so for the dilation morph. A better approach to morphing text may be to morph on a per-letter basis, rather than treating the whole text as a single shape. Some strategy would then have to be devised that determines which letter will morph to which, and how to deal with different Hausdorff distances between the letter pairs.

Both the Voronoi morph and the mixed morph often have small parts separating, moving, and then merging somewhere else (for example, the beak in the bird-to-ostrich morphs on <https://hausdorff-morphing.github.io>). Such artifacts may be circumvented by choosing a slightly warped Voronoi diagram, but this upsets the simplicity of the current methods. We can sometimes notice in the animations that the mixed morph is indeed not Lipschitz continuous, but since φ is rather small, this does not show clearly.

6 Conclusion

We introduced a new abstract morphing method based on Voronoi diagrams. This new method satisfies the same bounds on the Hausdorff distance as the previously introduced dilation morph, and is also 1-Lipschitz continuous. We have shown experimentally that the intermediate shapes of the Voronoi morph have areas that more closely match those of the input shapes than the dilation morph, but tends to have a perimeter that is larger than desired. To remedy this, we introduced a variant morph, the mixed morph, that we experimentally show to reduce this problem of increasing the perimeter. This mixed morph still satisfies the bounds on the Hausdorff distance, but is no longer 1-Lipschitz continuous. Our experimental analysis is the first we are aware of that analyses the development of area, perimeter, number of components and number of holes throughout the morphs.

An interesting open question is whether we can prevent the increase in perimeter caused by the Voronoi morph without losing 1-Lipschitz continuity. This would require somehow anticipating the moment when two pieces of boundary will meet, and smoothly bridging the gap between them over time, instead of just instantly filling it. To optimise the mixed morph, we can study the effects of choosing different φ , or even changing φ throughout the morph. Another direction is to develop other morphs that guarantee a smooth change of some distance measure other than the Hausdorff distance; we noted that it is unclear how to employ the Fréchet distance for morphing in the presence of multiple components.

A more practically oriented direction for further research would be to develop a less naive method of filling gaps than the mixed morph. It does not necessarily make sense to use the same radius for the closing operator everywhere, which sometimes closes gaps that will be opened again. However, any adaptation of this type will disrupt the conceptual simplicity of the Voronoi and mixed morphs.

References


- 1 P. K. Agarwal, J. Pach, and M. Sharir. State of the union (of geometric objects). In J. Goodman, J. Pach, and R. Pollack, editors, *Surveys on Discrete and Computational Geometry*, volume 453 of *Contemporary Mathematics*, pages 9–48. American Mathematical Society, 2008.

- 2 A. B. Albu, T. Beugeling, and D. Laurendeau. A morphology-based approach for interslice interpolation of anatomical slices from volumetric images. *IEEE Transactions on Biomedical Engineering*, 55(8):2022–2038, 2008.
- 3 H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 13(3):251–265, 1995.
- 4 H. Alt, P. Braß, M. Godau, C. Knauer, and C. Wenk. Computing the Hausdorff distance of geometric patterns and shapes. In B. Aronov, S. Basu, J. Pach, and M. Sharir, editors, *Discrete and Computational Geometry - The Goodman-Pollack Festschrift*, pages 65–76. Springer, 2003.
- 5 H. Alt and L. J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In *Handbook of Computational Geometry*, pages 121–153. Elsevier, 2000.
- 6 N. Aspert, D. Santa-Cruz, and T. Ebrahimi. Mesh: Measuring errors between surfaces using the Hausdorff distance. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, volume 1, pages 705–708, 2002.
- 7 M. J. Atallah. A linear time algorithm for the Hausdorff distance between convex polygons. *Information Processing Letters*, 17(4):207–209, 1983.
- 8 Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Company, 2013.
- 9 G. Barequet, M. T. Goodrich, A. Levi-Steiner, and D. Steiner. Contour interpolation by straight skeletons. *Graphical Models*, 66(4):245–260, 2004.
- 10 G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. *Computer Vision and Image Understanding*, 63(2):251–272, 1996.
- 11 G. Barequet and A. Vaxman. Reconstruction of multi-label domains from partial planar cross-sections. *Computer Graphics Forum*, 28(5):1327–1337, 2009.
- 12 J.-D. Boissonnat. Shape reconstruction from planar cross sections. *Computer Vision, Graphics, and Image Processing*, 44(1):1–29, 1988.
- 13 Q. W. Bouts, I. Kostitsyna, M. van Kreveld, W. Meulemans, W. Sonke, and K. Verbeek. Mapping polygons to the grid with small Hausdorff and Fréchet distance. In *Proceedings of the 24th Annual European Symposium on Algorithms*, pages 22:1–22:16, 2016.
- 14 K. Buchin, M. Buchin, W. Meulemans, and B. Speckmann. Locally correct Fréchet matchings. *Computational Geometry*, 76:1–18, 2019.
- 15 K. Buchin, E. W. Chambers, T. Ophelders, and B. Speckmann. Fréchet isotopies to monotone curves. In *Proceedings of the 33rd European Workshop on Computational Geometry*, pages 41–44, 2017.
- 16 E. W. Chambers, D. Letscher, T. Ju, and L. Liu. Isotopic Fréchet distance. In *Proceedings of the 23rd Canadian Conference on Computational Geometry*, 2011.
- 17 P. Cignoni, C. Rocchini, and R. Scopigno. Metro: measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.
- 18 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.
- 19 M.-P. Dubuisson and A. K. Jain. A modified Hausdorff distance for object matching. In *Proceedings of 12th IEEE International Conference on Pattern Recognition*, volume 1, pages 566–568, 1994.
- 20 C. Gotsman and V. Surazhsky. Guaranteed intersection-free polygon morphing. *Computers & Graphics*, 25(1):67–75, 2001.
- 21 R. M. Haralick, S. R. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(4):532–550, 1987.
- 22 O. van Kaick, H. Zhang, G. Hamarneh, and D. Cohen-Or. A survey on shape correspondence. *Computer Graphics Forum*, 30(6):1681–1707, 2011.
- 23 M. van Kreveld, T. Miltzow, T. Ophelders, W. Sonke, and J. L. Vermeulen. Between shapes, using the Hausdorff distance. *Computational Geometry*, 100:101817, 2022.
- 24 Z. Liu, L. Zhou, H. Leung, and H. P.-H. Shum. High-quality compatible triangulations and their application in interactive animation. *Computers & Graphics*, 76:60–72, 2018.

74:16 Abstract Morphing Using the Hausdorff Distance and Voronoi Diagrams

- 25 A. Maheshwari, J.-R. Sack, and C. Scheffer. Approximating the integral Fréchet distance. *Computational Geometry*, 70:13–30, 2018.
- 26 G. Rote. Lexicographic Fréchet matchings. In *Proceedings of the 30th European Workshop on Computational Geometry*, 2014.

Average Sensitivity of the Knapsack Problem

Soh Kumabe 

The University of Tokyo, Japan

Yuichi Yoshida  

National Institute of Informatics, Tokyo, Japan

Abstract

In resource allocation, we often require that the output allocation of an algorithm is stable against input perturbation because frequent reallocation is costly and untrustworthy. Varma and Yoshida (SODA'21) formalized this requirement for algorithms as the notion of average sensitivity. Here, the average sensitivity of an algorithm on an input instance is, roughly speaking, the average size of the symmetric difference of the output for the instance and that for the instance with one item deleted, where the average is taken over the deleted item.

In this work, we consider the average sensitivity of the knapsack problem, a representative example of a resource allocation problem. We first show a $(1 - \epsilon)$ -approximation algorithm for the knapsack problem with average sensitivity $O(\epsilon^{-1} \log \epsilon^{-1})$. Then, we complement this result by showing that any $(1 - \epsilon)$ -approximation algorithm has average sensitivity $\Omega(\epsilon^{-1})$. As an application of our algorithm, we consider the incremental knapsack problem in the random-order setting, where the goal is to maintain a good solution while items arrive one by one in a random order. Specifically, we show that for any $\epsilon > 0$, there exists a $(1 - \epsilon)$ -approximation algorithm with amortized recourse $O(\epsilon^{-1} \log \epsilon^{-1})$ and amortized update time $O(\log n + f_\epsilon)$, where n is the total number of items and $f_\epsilon > 0$ is a value depending on ϵ .

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases Average Sensitivity, Knapsack Problem, FPRAS

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.75

Funding *Soh Kumabe*: Supported by JST, PRESTO Grant Number JPMJPR192B.

Yuichi Yoshida: Supported by JST, PRESTO Grant Number JPMJPR192B.

1 Introduction

1.1 Background and Motivation

The *knapsack problem* is one of the most fundamental models in *resource allocation*, which handles the selection of good candidates under a budget constraint. For example, it can model the hiring process of employees in a company and the selection process of government projects. The knapsack problem is formally defined as follows. The input is a pair (V, W) , where V is a set of n items, $W \in \mathbb{R}_{>0}$ is a weight limit, and each item $i \in V$ has a positive weight $w(i) \leq W$ and value $v(i)$. The goal of the problem is to find a set of items $S \subseteq V$ that maximizes the total value $\sum_{i \in S} v(i)$, subject to the weight constraint $\sum_{i \in S} w(i) \leq W$.

Sometimes, the information used to allocate resources is uncertain or outdated. For example, suppose that a satellite isolated from the Earth is taking actions. The satellite has a list of potential actions V , and each action has a fixed value $v(i)$ and fuel consumption $w(i)$. Some of the actions may be infeasible at the moment due to the satellite's conditions, such as the atmosphere or surrounding space debris. The satellite's objective is to find a combination of feasible actions with the highest possible total value without running out of W amount of fuel.



© Soh Kumabe and Yuichi Yoshida;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 75; pp. 75:1–75:14

Leibniz International Proceedings in Informatics



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

75:2 Average Sensitivity of the Knapsack Problem

The control room on the Earth wants to know the satellite’s action for future planning. Since direct communication to the satellite is not always possible, the control room would simulate the satellite’s decision process. However, the list V that the control room has as potential actions for the satellite may be different from the list that the satellite uses. Even in such a situation, the control room would like to predict the actual actions taken by the satellite with some accuracy. Such a purpose can be achieved by designing algorithms with small *average sensitivity* [24, 30].

The following definitions are necessary to formally define the average sensitivity. Let V be a finite set. Then, the *Hamming distance*¹ between two sets $S, S' \subseteq V$ is $|S \Delta S'|$, where $S \Delta S' = (S \setminus S') \cup (S' \setminus S)$ is the *symmetric difference*. For two probability distributions \mathcal{S} and \mathcal{S}' over sets in V , the *earth mover’s distance* $\text{EM}(\mathcal{S}, \mathcal{S}')$ between them is given by

$$\min_{\mathcal{D}} \mathbb{E}_{(S, S') \sim \mathcal{D}} |S \Delta S'|, \quad (1)$$

where the minimum is taken over distributions of a pair of sets such that its marginal distributions on the first and the second coordinates are equal to \mathcal{S} and \mathcal{S}' , respectively. Let \mathcal{A} be a (randomized) algorithm for the knapsack problem, and let (V, W) be an instance of the knapsack problem. We abuse the notation \mathcal{A} (resp., \mathcal{A}^i) to represent the output (distribution) of the algorithm \mathcal{A} on the instance (V, W) (resp., $(V \setminus \{i\}, W)$). Then, the *average sensitivity* of the algorithm (on the instance (V, W)) is

$$\frac{1}{n} \sum_{i \in V} \text{EM}(\mathcal{A}, \mathcal{A}^i). \quad (2)$$

An algorithm is informally called *stable-on-average* if its average sensitivity is small.

1.2 Our Contributions

Our main contribution is a fully polynomial-time randomized approximation scheme (FPRAS) for the knapsack problem with a small average sensitivity:

► **Theorem 1.** *For any $\epsilon > 0$, there is a $(1 - \epsilon)$ -approximation algorithm for the knapsack problem with time complexity polynomial in $\epsilon^{-1}n$ and average sensitivity $O(\epsilon^{-1} \log \epsilon^{-1})$, where n is the number of items.*

It is noteworthy that Kumabe and Yoshida [20] have presented a $(1 - \epsilon)$ -approximation algorithm with $O(\epsilon^{-1} \log^3(nW))$ average sensitivity for the knapsack problem with integer weights. In contrast, our algorithm can be applied to an instance with non-integer weight and has a smaller average sensitivity bound independent of the weight limit W .

The following proposition states that the algorithm that outputs an optimal solution has an unbounded average sensitivity. Thus to bound the average sensitivity, we should allow for approximation as in Theorem 1.

► **Proposition 2.** *The average sensitivity of the algorithm that outputs an optimal solution can be as large as $\Omega(n)$.*

Let $\text{opt}(V)$ be the optimal value of the original instance. The basic idea behind the algorithm of Theorem 1 is classifying items into two categories according to whether their values are more than a threshold $\approx \epsilon \cdot \text{opt}(V)$. It is not difficult to design stable-on-average

¹ The Hamming distance is usually defined for strings, but our definition matches the formal definition by considering a set $S \subseteq V$ as a binary string of length $|V|$.

$(1 - \epsilon)$ -approximation algorithms when all the items belong to one of the two categories. Indeed, if all the items are small, i.e., with values at most $\epsilon \cdot \text{opt}(V)$, then we can prove that a variant of the greedy algorithm has an approximation ratio $(1 - O(\epsilon))$ and an $O(\epsilon^{-1})$ average sensitivity. If all items are large, i.e., with values at least $\epsilon \cdot \text{opt}(V)$, then any algorithm has a small average sensitivity because any feasible solution contains at most ϵ^{-1} items. To combine these two algorithms, we add a subset of the large items selected by the *exponential mechanism* [23] to the output. However, if we apply the exponential mechanism to all possible sets of large items, the average sensitivity becomes too high because the number of such sets is exponentially large. Therefore, we need to reduce the number of candidate sets appropriately without sacrificing the approximation guarantee. A more detailed overview is provided in Section 4.1.

Next, we show that our upper bound on the average sensitivity is tight up to a logarithmic factor in ϵ^{-1} .

► **Theorem 3.** *Let $\epsilon > 0$. Then any $(1 - \epsilon)$ -approximation algorithm for the knapsack problem has average sensitivity $\Omega(\epsilon^{-1})$.*

The *simple knapsack problem* is a special case of the general knapsack problem in which the value of each item is proportional to its weight. For this problem, we obtain a deterministic algorithm with a better average sensitivity:

► **Theorem 4.** *For any $\epsilon > 0$, there is a deterministic $(1 - \epsilon)$ -approximation algorithm for the simple knapsack problem with time complexity polynomial in $\epsilon^{-1}n$ and average sensitivity $O(\epsilon^{-1})$, where n is the number of items.*

Finally, we discuss the connection to dynamic algorithms. In the *incremental dynamic knapsack problem*, items arrive one by one, and the goal is to maintain an approximate solution for the current set of items. The amortized recourse of a dynamic algorithm is the average Hamming distance between the outputs before and after an item arrives. More formally, the *amortized recourse* of a deterministic algorithm over a stream v_1, \dots, v_n of items is defined as $\frac{1}{n} \sum_{i=1}^n |X_{i-1} \Delta X_i|$, where X_i is the solution of the algorithm right after v_i is added. The amortized recourse of a randomized algorithm is the expectation of amortized recourse over the randomness of the algorithm. When the arrival order is random, we can construct an algorithm for the incremental dynamic knapsack problem using our stable-on-average algorithm:

► **Theorem 5.** *For any $\epsilon > 0$, there exists $f_\epsilon > 0$ such that there is a $(1 - \epsilon)$ -approximation algorithm with amortized recourse $O(\epsilon^{-1} \log \epsilon^{-1})$ and update time $O(f_\epsilon + \log n)$ for the incremental knapsack problem in the random-order setting.*

The fact that the output of our stable-on-average algorithm does not depend on the arrival order of the items but depends only on the current set of items implies that this result can also be applied to the *decremental knapsack problem*, in which we are to maintain a good solution while the items are removed one by one from the initial set of items.

1.3 Related Work

1.3.1 Knapsack Problem

The *knapsack problem* is one of the 21 problems that was first proved to be NP-hard by Karp [16], but admits a pseudo-polynomial time algorithm via dynamic programming [19]. Ibarra and Kim [13] proposed the first fully polynomial-time approximation scheme (FPTAS)

for the knapsack problem. The main idea is to round the values of items into multiples of a small value, and run a dynamic programming algorithm on the resulting instance. Since then, several faster algorithms have been developed [5, 15, 17, 18, 21, 27].

The knapsack problem has been studied in the context of online settings. In the most basic *online knapsack problem*, items arrive one by one, and when an item arrives, it is irrevocably added to the knapsack or discarded. The difference from our dynamic model is that, in the online knapsack problem, once an item is accepted or rejected, the decision cannot be reverted. However, this is not the case in our model. While Marchetti-Spaccamela and Vercellis [22] showed that no algorithm has a bounded competitive ratio for this problem in general, Buchbinder and Naor [4] proposed an $O(\log(U/L))$ -competitive algorithm for the case where all weights are much smaller than the weight limit, where U and L are the upper and lower bounds on the *efficiency* $w(i)/v(i)$. Zhou, Chakrabarty, and Lukose showed that this bound is tight [32]. Several variations of the online knapsack problem have been investigated, such as the *removable online knapsack problem* [14], which allows removing items added from the knapsack, the *knapsack secretary problem* [2], which concerns items arriving in random order, and the *online knapsack problem with a resource buffer* [12], in which we have a buffer that can contain higher weights of items than the weight limit, and at the end, we can select a subset of the stored items as the output.

The problem most closely related to ours is the *dynamic setting*. In this problem, the goal is to maintain a good solution while items are added or deleted dynamically. In this model, there is no restriction on adding items that were previously deleted from the knapsack (and vice versa). While designing fast algorithms that maintain an exact solution is a major issue of focus in the context of data structures [1], the problem of maintaining approximate solutions has also been investigated in a variety of problems including the shortest path problem [28, 29], maximum matching problem [11, 25], and set cover problem [9]. Recently, dynamic approximation algorithms with bounded recourse have been studied for several problems including the bin-packing problem [8], set cover problem [9], and submodular cover problem [10]. As for the knapsack problem, a recent work of Böhm et al. [3] presented an algorithm that maintains a $(1 - \epsilon)$ -approximate solution with polylogarithmic update time in the fully dynamic setting, where both additions and deletions of items are allowed. Because this model considers both additions and deletions, the solution must be drastically changed even after a single update operation. Therefore, they proposed an algorithm that maintains a data structure from which a good solution can be recovered (with a time proportional to the size of the output), rather than explicitly maintaining the solution as a set of items.

1.3.2 Average Sensitivity

Murai and Yoshida [24] introduced the notion of average sensitivity for centralities, i.e., importance of nodes and edges, on networks to compare various notions of centralities. The notion of average sensitivity for graph problems was recently introduced by Varma and Yoshida [30], in which they studied various graph problems, including the minimum spanning tree problem, minimum cut problem, maximum matching problem, and minimum vertex cover problem. Zhou and Yoshida [31] presented a $(1 - \epsilon)$ -approximation algorithm for the maximum matching problem with sensitivity solely depending on ϵ , where the (*worst-case*) *sensitivity* is defined as (2) with the average replaced by the maximum over i . Kumabe and Yoshida [20] presented a stable-on-average algorithm for the maximum weight chain problem on directed acyclic graphs. The approximation ratio of their algorithm is $(1 - \epsilon)$ and average sensitivity is polylogarithmic in the number of vertices in the graph, which roughly corresponds to the number of states of the dynamic programming. They designed

stable-on-average algorithms for a wide range of dynamic programming problems, including the knapsack problem with integer weights, by reducing them to the maximum weight chain problem. Peng and Yoshida [26] analyzed the average sensitivity of spectral clustering and showed that it is proportional to λ_2/λ_3^2 , where λ_i is the i -th smallest eigenvalue of the (normalized) Laplacian of the input graph. Intuitively, this bound indicates that spectral clustering is stable on average when the input graph can be well partitioned into two communities but not into three. This implies that spectral clustering is stable on average at relevant instances. The relation between *Differential privacy*, proposed by Dwork *et al.* [7], is given in the full version.

1.4 Organization

The rest of this paper is organized as follows. In Section 2, we review the basics of the knapsack problem, especially focusing on the fractional knapsack problem. In Section 3, we present a stable-on-average $(1 - O(\epsilon))$ -approximation algorithm when all items have value at most ϵ , where the analysis is given in the full version. In Section 4, we present an (inefficient) stable-on-average algorithm for the general setting, while some proofs are given in the full version. In the full version, we discuss on improving the time complexity to obtain an FPRAS, prove the $O(\epsilon^{-1})$ lower bound on the average sensitivity of $(1 - \epsilon)$ -approximation algorithms, present a deterministic $(1 - \epsilon)$ -approximation algorithm with $O(\epsilon^{-1})$ sensitivity for the simple knapsack problem, and discuss the application to the dynamic knapsack problem in the random-order setting.

2 Basic Facts about the Knapsack Problem

For an instance (V, W) of the knapsack problem, let $\text{opt}(V, W)$ be the optimal value for the instance. As W is often set to 1, we define $\text{opt}(V) := \text{opt}(V, 1)$ for convenience. In this work, we assume that each item has a unique (comparable) identifier that remains unchanged with the deletion of other items. This naturally defines the ordering of items. In our algorithms, we implicitly use this ordering for the tiebreak. For example, when we sort items, we use a sorting algorithm that is stable with respect to this ordering. For a set $S \subseteq V$, let $v(S)$ and $w(S)$ denote $\sum_{i \in S} v(i)$ and $\sum_{i \in S} w(i)$, respectively.

We consider the following fractional relaxation of the knapsack problem, which we call the *fractional knapsack problem*:

$$\begin{aligned} & \text{maximize} && \sum_{i \in V} v(i)x(i), \\ & \text{subject to} && \sum_{i \in V} w(i)x(i) \leq W, \\ & && 0 \leq x(i) \leq 1 \quad (i \in V). \end{aligned}$$

Then, we denote the optimal value of this problem by $\text{fopt}(V, W)$. We also define $\text{fopt}(V) := \text{fopt}(V, 1)$, and the *efficiency* of an item i is $v(i)/w(i)$. The following is well known:

► **Lemma 6** ([6]). *Suppose that items in V are sorted in non-increasing order of efficiency, i.e., $v(1)/w(1) \geq \dots \geq v(n)/w(n)$. Let k be the largest index with $w(1) + \dots + w(k) \leq 1$. Then, $\text{fopt}(V, 1)$ is achieved by the solution*

$$x(i) = \begin{cases} 1 & (i = 1, \dots, k), \\ \frac{1 - \sum_{j=1}^k w(j)}{w(i+1)} & (i = k + 1), \\ 0 & (i \geq k + 2). \end{cases}$$

75:6 Average Sensitivity of the Knapsack Problem

■ **Algorithm 1** A $(1 - O(\epsilon))$ -approximation algorithm with small average sensitivity.

```
1 Procedure MODIFIEDGREEDY( $\epsilon, V$ )
2   Reorder the items of  $V$  so that  $v(1)/w(1) \geq \dots \geq v(n)/w(n)$ ;
3   Let  $W$  be sampled from  $[1 - \epsilon, 1]$  uniformly at random;
4   Let  $t$  be the maximum index with  $\sum_{i=1}^t w(i) \leq W$ ;
5   return  $\{1, \dots, t\}$ .
```

Using Lemma 6, $\text{fopt}(V, W)$ can be computed in polynomial time. The following is a direct consequence of Lemma 6:

► **Lemma 7.** For $W \leq 1$, we have $\text{fopt}(V, W) \geq W \cdot \text{fopt}(V)$.

We give the proofs in the following two lemmas in full version.

► **Lemma 8.** $\text{opt}(V) \leq \text{fopt}(V) \leq 2\text{opt}(V)$ holds.

► **Lemma 9.** For an item set U and weight limit W , we have

$$\sum_{i \in U} (\text{fopt}(U, W) - \text{fopt}(U \setminus \{i\}, W)) \leq \text{fopt}(U, W).$$

3 Items with Small Values

In this section, we provide a stable-on-average algorithm for instances with each item having a small value. Specifically, we show the following:

► **Theorem 10.** For any $\epsilon, \delta > 0$, there exists an algorithm for the knapsack problem with average sensitivity $O(\epsilon^{-1})$ that, given an instance (V, W) with each item having value at most δ , outputs a solution with value at least $(1 - \epsilon)\text{fopt}(V, W) - \delta$.

This algorithm will be used in our algorithm for the general case as a subroutine in Section 4. Note that it suffices to design an algorithm for the case in which the weight limit is 1 because we can first divide the weight of each item by W and feed the resulting instance to the algorithm. Hence in the rest of this paper, we assume that the weight limit is 1 (and use the symbol W for a different purpose). A pseudocode of our algorithm is given in Algorithm 1. The proof of Theorem 10 is given in the full version.

4 General Case

In this section, we consider the general case of the knapsack problem, and prove the following:

► **Theorem 11.** For any $\epsilon > 0$, there exists a $(1 - \epsilon)$ -approximation algorithm for the knapsack problem with average sensitivity $O(\epsilon^{-1} \log \epsilon^{-1})$.

Note that this algorithm takes exponential time. We will discuss on improving time complexity in the full version.

4.1 Technical Overview and Algorithm Description

First, we explain the intuition behind our algorithm (Algorithm 2). As in Section 3, we assume that the weight limit is 1 without loss of generality. We fix a parameter $0 < \epsilon < 0.05$. We say that an item is *large* if its value is at least $\epsilon \cdot \text{fopt}(V)$ and *small* otherwise. If all items are small, MODIFIEDGREEDY (Algorithm 1) with the parameter ϵ has an approximation ratio $1 - O(\epsilon)$ and average sensitivity $O(\epsilon^{-1})$, and we are done. On the contrary, if all items are large, the procedure of outputting the optimal solution has average sensitivity $O(\epsilon^{-1})$ because any feasible solution has cardinality at most ϵ^{-1} . We combine these two observations to obtain an algorithm for the general case as follows. Note that here we don't concern about the running time; we accelerate it in the full version.

Let $L \subseteq V$ be the set of large items. First, we take a subset $R \subseteq L$ with $w(R) \leq 1$ that maximizes $v(R) + \text{opt}(V \setminus L, 1 - w(R))$. (Note that in this section, we do not take efficiency into consideration and therefore we assume that $\text{opt}(V \setminus L, 1 - w(R))$ can be computed.) Then, we output $R \cup \text{MODIFIEDGREEDY}(\epsilon, V \setminus L, 1 - w(R))$, where $\text{MODIFIEDGREEDY}(\epsilon, U, W)$ runs $\text{MODIFIEDGREEDY}(\epsilon, U)$ after replacing $w(i)$ with $w(i)/W$ for each $i \in U$.

However, this algorithm has the following two issues.

1. If the value of $\text{fopt}(V)$ changes upon deleting an item, the set L may drastically change. For example, consider the case when there are many items with values slightly less than $\epsilon \cdot \text{fopt}(V)$. Then, the set of small items added to the output may drastically change.
2. Even when the value $\text{fopt}(V)$ does not change, if the choice of R changes upon deleting an item, the value of $1 - w(R)$ also changes. Thus, the set of small items added to the output may drastically change.

To resolve the first issue, we sample a value threshold $c = O(\epsilon \cdot \text{fopt}(V))$ that classifies items as large or small from an appropriate distribution, instead of fixing it to $\epsilon \cdot \text{fopt}(V)$.

Now, we consider the second issue. Suppose we have sampled the same value threshold c both before and after deleting an item $i \in V$. There are two cases in which R changes after deleting the item i .

1. If the item i is large and $i \in R$, then the algorithm should change the choice of R because the item i would no longer exist.
2. If the item i is small, then the algorithm may change the choice of R because the value $\text{fopt}(V \setminus L, 1 - w(R))$ (for the original R) may decrease.

The first case is easy to resolve; R contains $O(\epsilon^{-1})$ many items and therefore, by taking average over i , this case contributes to the average sensitivity by $O(\epsilon^{-1})$.

To address the second case (deleting small items), we ensure that the distribution of R does not change significantly with small decreases in $\text{fopt}(V \setminus L, 1 - w(R))$. To this end, instead of considering the R with the maximum value of $v(R) + \text{fopt}(V \setminus L, 1 - w(R))$ among $R \subseteq L$, we apply the exponential mechanism [23]. Specifically, we sample R with probability proportional to the exponential of this value with appropriate scaling and rounding (see Line 13 in Algorithm 2). To ensure that the mechanism outputs a $(1 - \epsilon)$ -approximate solution, we reduce the number of candidates that can possibly be R to a constant. This is implemented by taking only one candidate A_t from the family of sets R with $w(R) \in [tc, (t + 1)c]$ for each integer t (see Line 7 in Algorithm 2). For technical reasons, we apply an exponential mechanism for the value $tc + \text{fopt}(V \setminus L, 1 - w(A_t))$, rather than the exact value $v(A_t) + \text{opt}(V \setminus L, 1 - w(A_t))$ (see Line 9).

In Sections 4.2 and 4.3, we analyze the approximation ratio and average sensitivity of Algorithm 2, respectively.

■ **Algorithm 2** A $(1 - O(\epsilon))$ -approximation algorithm with small average sensitivity.

```

1 Procedure STABLEONAVERAGEKNAPSACK( $\epsilon, V$ )
2   Sample  $c$  from the uniform distribution over  $[\epsilon \cdot \text{fopt}(V), 2\epsilon \cdot \text{fopt}(V)]$ ;
3   Let  $L \subseteq V$  be the set of items with value at least  $c$ ;
4   Let  $l = \lfloor \text{fopt}(V)/c \rfloor$ ;
5   for  $t = 0, \dots, l$  do
6     if there is a subset of  $L$  with value in  $[tc, (t+1)c]$  then
7       Let  $A_t \subseteq L$  be the set of items with smallest weight that satisfies
8          $tc \leq v(A_t) < (t+1)c$ ;
9         (if there are multiple choices, choose the lexicographically smallest one);
10      Let  $x_t = tc + \text{fopt}(V \setminus L, 1 - w(A_t))$ ;
11    else
12      Let  $A_t = \emptyset$  and  $x_t = -\infty$ ;
13  Let  $d = \frac{c}{10 \log(\epsilon^{-1})} = O\left(\frac{\epsilon}{\log(\epsilon^{-1})} \text{fopt}(V)\right)$ ;
14  Sample  $t^\circ \in \{0, \dots, l\}$  with probability proportional to  $\exp(x_{t^\circ}/d)$  and let
15     $R = A_{t^\circ}$ ;
16  return  $R \cup \text{MODIFIEDGREEDY}(V \setminus L, 1 - w(A_t))$ ;
```

4.2 Approximation Ratio

First, we analyze the approximation ratio of Algorithm 2. Let S^* be a set of items that attains $\text{opt}(V)$. Let $t^* \geq 0$ be an integer such that $v(S^* \cap L) \in [t^*c, (t^* + 1)c)$.

The following lemma bounds the loss in the approximation ratio caused by considering only A_0, \dots, A_l instead of all subsets of L as candidate sets that can possibly be R .

► **Lemma 12.** $x_{t^*} \geq (1 - 4\epsilon) \text{opt}(V)$ holds.

Proof. We have

$$\begin{aligned} x_{t^*} &= t^*c + \text{fopt}(V \setminus L, 1 - w(A_{t^*})) \geq t^*c + \text{fopt}(V \setminus L, 1 - w(S^* \cap L)) \\ &\geq t^*c + v(S^* \setminus L) \geq v(S^* \cap L) - c + v(S^* \setminus L) = \text{opt}(V) - c \geq (1 - 4\epsilon) \text{opt}(V), \end{aligned}$$

where the first equality is from the definition of x_{t^*} , the first inequality is from $w(A_{t^*}) \leq w(S^* \cap L)$, which is ensured by Line 7 in the algorithm, the second inequality is from $\text{fopt}(V \setminus L, 1 - w(S^* \cap L)) \geq \text{opt}(V \setminus L, 1 - w(S^* \cap L)) = v(S^* \setminus L)$, the third inequality is from the definition of t^* , the second equality is from the optimality of S^* , and the last inequality is from $c \leq 2\epsilon \cdot \text{fopt}(V) \leq 4\epsilon \cdot \text{opt}(V)$. ◀

Next we analyze the loss in the approximation ratio caused by the exponential method applied at Line 13. We prove the following.

► **Lemma 13.** $\mathbb{E}[x_{t^\circ}] \geq (1 - 3\epsilon)x_{t^*}$ holds.

Proof. We have

$$\begin{aligned}
\Pr[x_{t^\circ} \leq (1 - \epsilon)x_{t^*}] &= \frac{\sum_{t \in \{0, \dots, l\} : x_t \leq (1 - \epsilon)x_{t^*}} \exp\left(\frac{x_t}{d}\right)}{\sum_{t \in \{0, \dots, l\}} \exp\left(\frac{x_t}{d}\right)} \leq \frac{(l + 1) \exp\left(\frac{(1 - \epsilon)x_{t^*}}{d}\right)}{\exp\left(\frac{x_{t^*}}{d}\right)} \\
&= (l + 1) \exp\left(-\frac{\epsilon x_{t^*}}{d}\right) \\
&\leq (l + 1) \exp\left(-\frac{\epsilon(1 - 4\epsilon)\text{opt}(V)}{d}\right) \leq (l + 1) \exp\left(-\frac{5}{2} \log \epsilon^{-1} (1 - 4\epsilon)\right) \\
&= (l + 1) \epsilon^{\frac{5}{2}(1 - 4\epsilon)} \leq 2\epsilon^{-1} \cdot \epsilon^2 = 2\epsilon. \tag{3}
\end{aligned}$$

Here, the first equality is from Line 13 of the algorithm, the first inequality is from the fact that there are at most $l + 1$ indices t with $x_t \leq (1 - \epsilon)x_{t^*}$, the second inequality is from Lemma 12, the third inequality is from

$$d = \frac{c}{10 \log \epsilon^{-1}} \leq \frac{\epsilon}{5 \log \epsilon^{-1}} \cdot \text{fopt}(V) \leq \frac{5}{2} \cdot \frac{\epsilon}{\log \epsilon^{-1}} \cdot \text{opt}(V),$$

and the last inequality is from $l + 1 \leq \epsilon^{-1} + 1 \leq 2\epsilon^{-1}$ and $\frac{5}{2}(1 - 4\epsilon) \geq 2$, which is from $\epsilon \leq 0.05$. Therefore, we have

$$\mathbb{E}[x_{t^\circ}] \geq (1 - \epsilon)x_{t^*} \Pr[x_{t^\circ} > (1 - \epsilon)x_{t^*}] \geq (1 - \epsilon)(1 - 2\epsilon)x_{t^*} \geq (1 - 3\epsilon)x_{t^*}.$$

Here, the second inequality is from (3). \blacktriangleleft

Combining the lemmas above yields the following.

► Lemma 14. $\mathbb{E}[v(A_{t^\circ}) + \text{fopt}(V \setminus L, 1 - w(A_{t^\circ}))] \geq (1 - 7\epsilon)\text{opt}(V)$ holds.

Proof. We have

$$\begin{aligned}
&\mathbb{E}[v(A_{t^\circ}) + \text{fopt}(V \setminus L, 1 - w(A_{t^\circ}))] \\
&\geq \mathbb{E}[x_{t^\circ}] \geq (1 - 3\epsilon)x_{t^*} \geq (1 - 3\epsilon)(1 - 4\epsilon)\text{opt}(V) \geq (1 - 7\epsilon)\text{opt}(V),
\end{aligned}$$

where the first inequality is from Line 9 of the algorithm, the second inequality is from Lemma 13, and the third inequality is from Lemma 12. \blacktriangleleft

Now we bound the approximation ratio of Algorithm 2. Let $\mathcal{A}_{\text{SMALL}}(V', W')$ be the output of Algorithm 1 on V' , where all weights in the input are divided by W' .

► Lemma 15. *The approximation ratio of Algorithm 2 is at least $1 - 12\epsilon$.*

Proof. Let \mathcal{A} be the output distribution of Algorithm 2 applied on the instance $(V, 1)$. We have

$$\begin{aligned}
\mathbb{E}[\mathcal{A}] &= \mathbb{E}[v(A_{t^\circ}) + v(\mathcal{A}_{\text{SMALL}}(V \setminus L, 1 - w(A_{t^\circ})))] \\
&\geq \mathbb{E}[v(A_{t^\circ}) + (1 - \epsilon)\text{fopt}(V \setminus L, 1 - w(A_{t^\circ})) - c] \\
&\geq \mathbb{E}[(1 - \epsilon)(v(A_{t^\circ}) + \text{fopt}(V \setminus L, 1 - w(A_{t^\circ}))) - c] \\
&\geq \mathbb{E}[(1 - \epsilon)(1 - 7\epsilon)\text{opt}(V) - c] \geq (1 - 12\epsilon)\text{opt}(V),
\end{aligned}$$

where the first inequality is from the analysis of Algorithm 1 given in the full version, the third inequality is from Lemma 14, and the last inequality is from $c \leq 2\text{fopt}(V) \leq 4\text{opt}(V)$. \blacktriangleleft

4.3 Average Sensitivity

In this section, we discuss bounding the average sensitivity of Algorithm 2. For the parameters used in the algorithm applied on the instance $(V \setminus \{i\}, 1)$, we use symbols c^i, d^i , and R^i to denote c, d , and R , respectively (and use c, d , and R for the instance $(V, 1)$). Similarly, we use the symbols A_t^i and x_t^i for $t = 0, \dots, l$ to denote A_t and x_t , respectively, for the instance $(V \setminus \{i\}, 1)$. We first prove that the distributions of R and R^i are sufficiently close on average, where the average is taken over $i \in V$ (Lemma 22). Subsequently, we combine the analysis for large and small items (Lemma 24).

The distributions of R and R^i may differ for the following two reasons: the difference between the distributions of c and c^i , and the absence of the item i in the instance $V \setminus \{i\}$. We first focus on the second reason. Specifically, we fix the value \hat{c} and assume $c = c^i = \hat{c}$. In this case, we prove that the distribution of R is sufficiently close to that of R^i on average, where the average is taken over $i \in V$. To this end, we analyze the following quantity:

$$\sum_{i \in V} \sum_{A \subseteq V} \max(0, \Pr[R = A \mid c = \hat{c}] - \Pr[R^i = A \mid c^i = \hat{c}]). \quad (4)$$

Because $\Pr[R = A \mid c = \hat{c}]$ is positive only if $A = A_t$ for some $t \in \{0, \dots, l\}$, we have

$$(4) = \sum_{i \in V} \sum_{t \in \{0, \dots, l\}} \max(0, \Pr[R = A_t \mid c = \hat{c}] - \Pr[R^i = A_t \mid c^i = \hat{c}]). \quad (5)$$

We analyze the sum in (5) by dividing it into two cases: $i \in L$ and $i \in V \setminus L$. We first show that $x_t \geq x_t^i$ holds for all i and t .

► **Lemma 16.** *For any $t \in \{0, \dots, l\}$ and $i \in V$, we have $x_t \geq x_t^i$.*

Proof. If $x_t = -\infty$, then we have $x_t^i = -\infty$ due to Line 6 of Algorithm 2. Hereafter, we assume $x_t \neq -\infty$. We prove the lemma by considering the following three cases:

▷ **Claim 17.** If $i \in L \setminus A_t$, we have $x_t = x_t^i$.

▷ **Claim 18.** If $i \in A_t$, we have $x_t \geq x_t^i$.

▷ **Claim 19.** If $i \in V \setminus L$, we have $x_t \geq x_t^i$.

We give the proofs of these claims in the full version. Then we complete the case analysis and the lemma is proved. ◀

The next lemma handles the case $i \in L$. We give the proof in the full version.

► **Lemma 20.** *For any $\hat{c} \in \mathbb{R}$, we have*

$$\sum_{i \in L} \sum_{t \in \{0, \dots, l\}} \max(0, \Pr[R = A_t \mid c = \hat{c}] - \Pr[R^i = A_t \mid c^i = \hat{c}]) \leq \epsilon^{-1}.$$

Now we consider the case $i \in V \setminus L$.

► **Lemma 21.** *For any $\hat{c} \in \mathbb{R}$, we have*

$$\sum_{i \in V \setminus L} \sum_{t \in \{0, \dots, l\}} \max(0, \Pr[R = A_t \mid c = \hat{c}] - \Pr[R^i = A_t \mid c^i = \hat{c}]) \leq 10\epsilon^{-1} \log \epsilon^{-1}.$$

Proof. Let $\hat{d} = \frac{\hat{c}}{10 \log(\epsilon^{-1})}$. Then, we have

$$\begin{aligned}
& \Pr[R = A_t \mid c = \hat{c}] - \Pr[R^i = A_t \mid c^i = \hat{c}] \\
&= \frac{\exp(x_t/\hat{d})}{\sum_{t \in \{0, \dots, l\}} \exp(x_t/\hat{d})} - \frac{\exp(x_t^i/\hat{d})}{\sum_{t \in \{0, \dots, l\}} \exp(x_t^i/\hat{d})} \leq \frac{\exp(x_t/\hat{d}) - \exp(x_t^i/\hat{d})}{\sum_{t \in \{0, \dots, l\}} \exp(x_t/\hat{d})} \\
&= \left(1 - \exp\left(-\frac{x_t - x_t^i}{\hat{d}}\right)\right) \frac{\exp(x_t/\hat{d})}{\sum_{t \in \{0, \dots, l\}} \exp(x_t/\hat{d})} \leq \frac{x_t - x_t^i}{\hat{d}} \Pr[R = A_t \mid c = \hat{c}]. \quad (6)
\end{aligned}$$

Here, the first equality is from Line 13 of Algorithm 2, the first inequality is from Lemma 16, and the last inequality is from $1 - \exp(-x) \leq x$. Now, we have

$$\begin{aligned}
& \sum_{i \in V \setminus L} \sum_{t \in \{0, \dots, l\}} \max(0, \Pr[R = A_t \mid c = \hat{c}] - \Pr[R^i = A_t \mid c^i = \hat{c}]) \\
&\leq \sum_{i \in V \setminus L} \sum_{t \in \{0, \dots, l\}} \frac{x_t - x_t^i}{\hat{d}} \cdot \Pr[R = A_t \mid c = \hat{c}] \\
&\leq \sum_{t \in \{0, \dots, l\}} \frac{\text{fopt}(V \setminus L, 1 - w(A_t))}{\hat{d}} \cdot \Pr[R = A_t \mid c = \hat{c}] \\
&\leq \frac{\text{fopt}(V)}{\hat{d}} \sum_{t \in \{0, \dots, l\}} \Pr[R = A_t \mid c = \hat{c}] \leq 10\epsilon^{-1} \log \epsilon^{-1}.
\end{aligned}$$

Here, the first inequality is obtained from (6) and Claim 19. The second inequality is obtained from:

$$\begin{aligned}
\sum_{i \in V \setminus L} (x_t - x_t^i) &= \sum_{i \in V \setminus L} (\text{fopt}(V \setminus L, 1 - w(A_t)) - \text{fopt}((V \setminus L) \setminus \{i\}, 1 - w(A_t))) \\
&\leq \text{fopt}(V \setminus L, 1 - w(A_t)),
\end{aligned}$$

which is obtained from Lemma 9. The last inequality is from Line 12 of Algorithm 2. ◀

Combining Lemmas 20 and 21, we obtain the following.

► **Lemma 22.** *We have*

$$\sum_{i \in V} \sum_{A \subseteq V} \max(0, \Pr[R = A \mid c = \hat{c}] - \Pr[R^i = A \mid c^i = \hat{c}]) \leq 11\epsilon^{-1} \log \epsilon^{-1}.$$

Proof. The claim immediately follows from Lemmas 20 and 21 and the fact that V is a disjoint union of $V \setminus L$ and L . ◀

Now, we evaluate the average sensitivity. Let \mathcal{A}^i be the output distribution of Algorithm 2 applied on the instance $V \setminus \{i\}$.

To bound the earth mover's distance, we consider transporting the probability mass in such a way the mass of \mathcal{A} corresponding to a particular choice of c is transported to that of \mathcal{A}^i for the same c^i (as far as we can). For the same choice of c and c^i , we consider transporting the probability mass in such a way that the mass corresponding to a particular choice of R is transported to that for the same R^i (as far as we can). For the same choice of c, R and c^i, R^i , we consider transporting the probability mass in a manner similar to the analysis of Algorithm 1. The remaining mass is transported arbitrarily.

First, we bound the contribution of the difference between the distributions of c and c^i to the earth mover's distance. Let f and f^i be the probability density functions of c and c^i , respectively. Precisely, $f(c) = \frac{1}{\epsilon \cdot \text{fopt}(V)}$ if $\epsilon \cdot \text{fopt}(V) \leq c \leq 2\epsilon \cdot \text{fopt}(V)$ and 0 otherwise. Similarly, $f^i(c^i) = \frac{1}{\epsilon \cdot \text{fopt}(V \setminus \{i\})}$ if $\epsilon \cdot \text{fopt}(V \setminus \{i\}) \leq c^i \leq 2\epsilon \cdot \text{fopt}(V \setminus \{i\})$ and 0 otherwise.

75:12 Average Sensitivity of the Knapsack Problem

► **Lemma 23.** *We have*

$$\sum_{i \in V} \int_{\epsilon \cdot \text{fopt}(V)}^{2\epsilon \cdot \text{fopt}(V)} \max(0, f(\hat{c}) - f^i(\hat{c})) \, d\hat{c} \leq 2.$$

Proof. We have

$$\begin{aligned} \sum_{i \in V} \int_{\epsilon \cdot \text{fopt}(V)}^{2\epsilon \cdot \text{fopt}(V)} \max(0, f(\hat{c}) - f^i(\hat{c})) \, d\hat{c} &= \sum_{i \in V} \int_{\max\{\epsilon \cdot \text{fopt}(V), 2\epsilon \cdot \text{fopt}(V \setminus \{i\})\}}^{2\epsilon \cdot \text{fopt}(V)} \frac{1}{\epsilon \cdot \text{fopt}(V)} \, d\hat{c} \\ &\leq \frac{1}{\epsilon \cdot \text{fopt}(V)} \sum_{i \in V} (2\epsilon \cdot (\text{fopt}(V) - \text{fopt}(V \setminus \{i\}))) \leq \frac{1}{\epsilon \cdot \text{fopt}(V)} \cdot 2\epsilon \cdot \text{fopt}(V) = 2, \end{aligned}$$

where the first inequality is from the fact that $f(\hat{c}) \leq f^i(\hat{c})$ holds if $\epsilon \cdot \text{fopt}(V) \leq \hat{c} \leq 2\epsilon \cdot \text{fopt}(V \setminus \{i\})$, and the last inequality is from Lemma 9. ◀

Finally, we bound the average sensitivity.

► **Lemma 24.** *The average sensitivity of Algorithm 2 is at most $12\epsilon^{-1} \log \epsilon^{-1}$.*

Proof. We have

$$\begin{aligned} &\frac{1}{n} \sum_{i \in V} \text{EM}(\mathcal{A}, \mathcal{A}^i) \\ &\leq \frac{1}{n} \sum_{i \in V} \int_{\epsilon \cdot \text{fopt}(V)}^{2\epsilon \cdot \text{fopt}(V)} \max(0, f(\hat{c}) - f^i(\hat{c})) \, d\hat{c} \cdot n \\ &\quad + \frac{1}{n} \sum_{i \in V} \int_{\epsilon \cdot \text{fopt}(V)}^{2\epsilon \cdot \text{fopt}(V)} \sum_{A \subseteq V} \left(\max(0, \Pr[R = A \mid c = \hat{c}] - \Pr[R^i = A \mid c^i = \hat{c}]) \cdot n \right. \\ &\quad \left. + \Pr[R = A \mid c = \hat{c}] \cdot \text{EM}(\mathcal{A}_{\text{SMALL}}(V \setminus L, 1 - w(A)), \mathcal{A}_{\text{SMALL}}((V \setminus L) \setminus \{i\}, 1 - w(A))) \right) f(\hat{c}) \, d\hat{c} \\ &\leq 2 + \frac{1}{n} \int_{\epsilon \cdot \text{fopt}(V)}^{2\epsilon \cdot \text{fopt}(V)} \left(11\epsilon^{-1} \log \epsilon^{-1} n + \sum_{i \in V} \sum_{A \subseteq V} \Pr[R = A \mid c = \hat{c}] (\epsilon^{-1} + 1) \right) f(\hat{c}) \, d\hat{c} \\ &= 2 + \frac{1}{n} \int_{\epsilon \cdot \text{fopt}(V)}^{2\epsilon \cdot \text{fopt}(V)} (11\epsilon^{-1} \log \epsilon^{-1} n + (\epsilon^{-1} + 1) \cdot n) f(\hat{c}) \, d\hat{c} \\ &= 2 + 11\epsilon^{-1} \log \epsilon^{-1} + \epsilon^{-1} + 1 \leq 12\epsilon^{-1} \log \epsilon^{-1}, \end{aligned}$$

where the first inequality is due to the transport of the probability mass, and the second inequality is from Lemmas 23, 22, and the analysis of Algorithm 1 given in the full version. ◀

Proof of Theorem 11. Applying Lemma 15 and Lemma 24 and replacing ϵ with $\min(0.05, \epsilon/12)$ proves this theorem. ◀

References

- 1 Mikhaïl Atallah and Marina Blanton. *Algorithms and theory of computation handbook*. CRC press, 2009.
- 2 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Approximation, randomization, and combinatorial optimization. Algorithms and techniques*, pages 16–28. Springer, 2007.
- 3 Martin Böhm, Franziska Eberle, Nicole Megow, Bertrand Simon, Lukas Nölke, Jens Schlöter, and Andreas Wiese. Fully dynamic algorithms for knapsack problems with polylogarithmic update time. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2021.

- 4 Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Mathematics of Operations Research*, 34(2):270–286, 2009.
- 5 Timothy Chan. Approximation schemes for 0-1 knapsack. In *Symposium on Simplicity in Algorithms*, 2018.
- 6 George Dantzig. Discrete variable extremum problems. *Operations Research*, 5:266–277, 1957.
- 7 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, pages 265–284. Springer, 2006.
- 8 Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-dynamic bin packing with little repacking. In *International Colloquium on Automata, Languages, and Programming*, 2018.
- 9 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *ACM SIGACT Symposium on Theory of Computing*, pages 537–550, 2017.
- 10 Anupam Gupta and Roie Levin. Fully-dynamic submodular cover with bounded recourse. *IEEE Symposium on Foundations of Computer Science*, pages 1147–1157, 2020.
- 11 Manoj Gupta and Richard Peng. Fully dynamic $(1 + \epsilon)$ -approximate matchings. In *IEEE Symposium on Foundations of Computer Science*, pages 548–557, 2013.
- 12 Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku. Online knapsack problems with a resource buffer. In *International Symposium on Algorithms and Computation*, 2019.
- 13 Oscar Ibarra and Chul Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.
- 14 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In *International Colloquium on Automata, Languages, and Programming*, pages 293–305, 2002.
- 15 Ce Jin. An improved fptas for 0-1 knapsack. In *International Colloquium on Automata, Languages, and Programming*, 2019.
- 16 Richard Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.
- 17 Hans Kellerer and Ulrich Pferschy. A new fully polynomial time approximation scheme for the knapsack problem. *Journal of Combinatorial Optimization*, 3(1):59–71, 1999.
- 18 Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an fptas for the knapsack problem. *Journal of Combinatorial Optimization*, 8(1):5–11, 2004.
- 19 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.
- 20 Soh Kumabe and Yuichi Yoshida. Average sensitivity of dynamic programming. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1925–1961, 2022.
- 21 Eugene Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.
- 22 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68(1):73–104, 1995.
- 23 Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *IEEE Symposium on Foundations of Computer Science*, pages 94–103, 2007.
- 24 Shogo Murai and Yuichi Yoshida. Sensitivity analysis of centralities on unweighted networks. In *The World Wide Web Conference*, pages 1332–1342, 2019.
- 25 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *ACM Symposium on Theory of Computing*, pages 457–464, 2010.
- 26 Pan Peng and Yuichi Yoshida. Average sensitivity of spectral clustering. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1132–1140. ACM, 2020.
- 27 Donguk Rhee. *Faster fully polynomial approximation schemes for knapsack problems*. PhD thesis, Massachusetts Institute of Technology, 2015.
- 28 Yossi Shiloach and Shimon Even. An on-line edge-deletion problem. *Journal of the ACM*, 28(1):1–4, 1981.

75:14 Average Sensitivity of the Knapsack Problem

- 29 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *IEEE Symposium on Foundations of Computer Science*, pages 436–455, 2019.
- 30 Nithin Varma and Yuichi Yoshida. Average sensitivity of graph algorithms. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 684–703, 2021.
- 31 Yuichi Yoshida and Samson Zhou. Sensitivity analysis of the maximum matching problem. In *Innovations in Theoretical Computer Science*, pages 58:1–58:20, 2021.
- 32 Yunhong Zhou, Deeparnab Chakrabarty, and Rajan Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In *International Workshop on Internet and Network Economics*, pages 566–576. Springer, 2008.

Cardinality Estimation Using Gumbel Distribution

Aleksander Łukasiewicz  

Faculty of Mathematics and Computer Science, University of Wrocław, Poland

Przemysław Uznański  

Faculty of Mathematics and Computer Science, University of Wrocław, Poland

Abstract

Cardinality estimation is the task of approximating the number of distinct elements in a large dataset with possibly repeating elements. **LogLog** and **HyperLogLog** (c.f. Durand and Flajolet [ESA 2003], Flajolet et al. [Discrete Math Theor. 2007]) are small space sketching schemes for cardinality estimation, which have both strong theoretical guarantees of performance and are highly effective in practice. This makes them a highly popular solution with many implementations in big-data systems (e.g. Algebird, Apache DataSketches, BigQuery, Presto and Redis). However, despite having simple and elegant formulation, both the analysis of **LogLog** and **HyperLogLog** are extremely involved – spanning over tens of pages of analytic combinatorics and complex function analysis.

We propose a modification to both **LogLog** and **HyperLogLog** that replaces discrete geometric distribution with the continuous Gumbel distribution. This leads to a very short, simple and elementary analysis of estimation guarantees, and smoother behavior of the estimator.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Streaming algorithms, Cardinality estimation, Sketching, Gumbel distribution

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.76

Related Version *Full Version*: <https://arxiv.org/abs/2008.07590>

Funding *Aleksander Łukasiewicz*: Polish National Science Centre grant 2019/33/B/ST6/00298.
Przemysław Uznański: Polish National Science Centre grant 2019/33/B/ST6/00298.

Acknowledgements We would like to thank Seth Pettie for useful remarks that helped us improve the paper.

1 Introduction

In the cardinality estimation problem we are presented with a dataset consisting of many items, and some of these items might appear more than once. Our goal is to process this dataset efficiently, in order to estimate the number n of *distinct* elements it contains. Here, efficiently means in small auxiliary space, and with fast processing time per each item. A natural scenario to consider is a *stream* processing of a dataset, with stream of events being either element *insertions* to the multiset or *queries* of the multiset cardinality.

A folklore information theoretic analysis reveals that this problem over universe of u elements requires at least u bits of memory to answer queries exactly. However, in many practical settings it is sufficient to provide an approximation of the actual cardinality. One of the possible real-world scenarios is a problem of estimating the number of unique addresses in packets that a router observes, in order to detect malicious behaviors and attacks. Here, the challenge arises from the limited computational capabilities of the router and sheer volume of the data that can be observed over e.g. a day.

The theoretical study of the *cardinality estimation* was initiated by the seminal work of Flajolet and Martin [20]. From that point, two separate lines of research follow. First, there has been a considerable effort put into developing approximation schemes with so called (ε, δ) -guarantees, meaning that they guarantee outputting $(1 + \varepsilon)$ -multiplicative approximation of



© Aleksander Łukasiewicz and Przemysław Uznański;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 76; pp. 76:1–76:13
Leibniz International Proceedings in Informatics



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

the cardinality, with probability at least $1 - \delta$. Here, we mention [6, 7, 8, 11, 22, 23, 29] on the upper-bound side and [6, 10, 27, 28, 36] on lower-bound side. The high-level takeaway message is that one can construct approximate schemes that provide $(1 + \varepsilon)$ -multiplicative approximation to the number of distinct elements, using an order of ε^{-2} space, and that this dependency on ε is tight. More specifically, the work of Błasiok [11] settles the bit-complexity of the problem, by providing $\mathcal{O}(\frac{\log \delta^{-1}}{\varepsilon^2} + \log n)$ bits of space upper-bound, and this complexity is optimal by a matching lower bound [28]. To achieve such small space usage, a number of issues have to be resolved, and a very sophisticated machinery of expanders and pseudo-randomness is deployed.

The other line of work is more practical in nature, and focuses on providing variance bounds for efficient algorithms. The bounds are usually of the form $\sim 1/\sqrt{k}$ where k is some measure of space-complexity of algorithms (usually, corresponds to the number of parallel estimation processes). This approach includes work of [9, 12, 14, 16, 18, 19, 21, 24, 30, 31, 34, 35]. Recently, Pettie and Wang started the study of the intrinsic tradeoff between the space complexity of the cardinality estimation sketch and its estimation error by introducing the notion of *memory-variance product* (MVP) [31]. They proposed a *Fishmonger* sketch that has an MVP equal to $H_0/I_0 \approx 1.98$ (where H_0, I_0 are some precisely defined constants) and they also proved that this is the best MVP that one can get in a class of *linearizable* sketches (in fact all the popular *mergeable* sketches are linearizable). In the very recent follow-up work Pettie, Wang and Yin studied the MVPs of non-mergeable sketches [32].

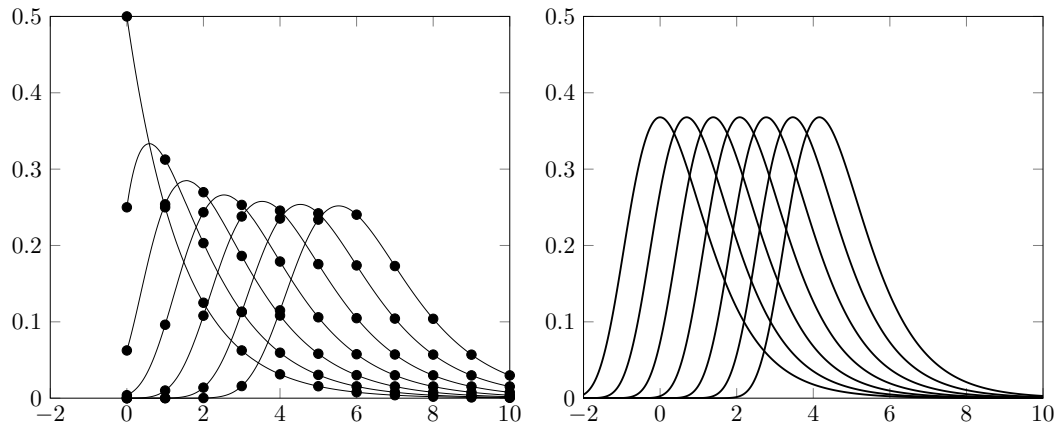
We now focus on two specific algorithms, namely **LogLog** [16] and its refined version called **HyperLogLog** [19]. The guarantees that these algorithms provide for variance are approximately $1.3/\sqrt{k}$ and $1.04/\sqrt{k}$ respectively, when using k integer registers. Both are based on a simple principle of observing the maximal number of trailing zeroes in the binary representation of hashes of elements in the stream, although they vary in the way they extract the final estimate from this observed value (we will discuss those details in the following section). In addition to being easy to state and provided with theoretical guarantees, they are highly practical in nature. We note the following works on algorithmic engineering of practical variants [17, 26, 37], with actual implementations e.g. in Algebird [1], BigQuery [2], Apache DataSketch [3], Presto [4] and Redis [5].

Despite its simplicity and popularity, **LogLog** and **HyperLogLog** are exceptionally tough to analyze. We note that both papers analyzing **LogLog** and later **HyperLogLog** use a heavy machinery of tools from analytic combinatorics and complex function analysis e.g. Mellin transform, poissonization and analytical depoissonization. In fact, unpacking the main tool used in the paper requires understanding of another tens of pages from [33]. Additionally, both papers are presented in a highly compressed form. Thus, the analysis is not easily digestible by a typical computer scientist, and has to be accepted “as it is” in a black-box manner, without actually unpacking it.

This creates an unsatisfactory situation where one of the most popular and most elegant algorithms for the cardinality estimation problem has to be treated as a black-box from the perspective of its performance guarantees. It is an obstacle both in terms of popularization of the **LogLog** and **HyperLogLog** algorithms, and in terms of scientific progress. We note that those algorithms are generally omitted during majority of theoretical courses on streaming and big data algorithms.

Our contribution: Gumbel distribution

Our contribution comes in two factors. First, we observe that the key part of **LogLog** and **HyperLogLog** algorithms is counting the trailing zeroes in the binary representation of a hash of element. This random variable is distributed according to a geometric distribution. Both



■ **Figure 1** Distribution of $\max\{X_1, \dots, X_k\}$ for $k \in \{1, 2, 4, 8, 16, 32, 64\}$ where X_i iid random variables distributed according to discrete Geometric distribution (on the left) and Gumbel distribution (on the right). Discrete distribution given by $f_k(x) = (1 - 2^{-x-1})^k - (1 - 2^{-x})^k$ is drawn with continuous intermediate values for smooth drawing.

LogLog and HyperLogLog estimate the cardinality using the maximum value of the count of trailing zeroes observed over all elements of the dataset. However, the distribution of the maximum of many discrete random variables drawn from an identical geometric distribution is not distributed according to a geometric distribution. This is unwieldy to handle in the analysis in [19].

We propose the following: as the first step we replace the discrete geometric distribution with its continuous counterpart, i.e. the exponential distribution with the CDF $1 - e^{-x}$. Next, we take a maximum of N independent repetitions of our algorithm which can be simulated by, e.g., replacing each update x with N updates of the form (x, i) for $i \in [N]$. This yields the CDF of the form $(1 - e^{-x})^N$. Intuitively, we expect this manipulation to have a smoothing effect on the irregularities of LogLog and HyperLogLog (which performance deteriorate greatly for very small values of n). Third and the final step is to take a limit of $N \rightarrow \infty$, while maintaining a proper normalization of the distribution (i.e., we take a shift by $\ln N$), resulting in a CDF of the form $F(X) = \lim_{N \rightarrow \infty} (1 - e^{-x - \ln N})^N$.

A little manipulation gives us $F(X) = \lim_{N \rightarrow \infty} (1 - \frac{e^{-x}}{N})^N = e^{-e^{-x}}$ which is precisely the CDF of the Gumbel distribution, with the following crucial property

If X_1, \dots, X_k are independent random variables drawn from a Gumbel distribution, then $Z = \max(X_1, \dots, X_k) - \ln(k)$ is also distributed according to the same Gumbel distribution.

This allows us to simplify extraction of the value of k from $\max(X_1, \dots, X_k)$, since we are always dealing with the same type of error (distributed according to the Gumbel distribution) on top of the value $\ln(k)$.

Our contribution: Simpler analysis

Our second contribution comes in the form of a simple analysis of the performance guarantees of the estimation. We note that since our observable can be interpreted as an observable from LogLog or HyperLogLog repeated N -times (for some very large value of N), we expect to get a similar type of the guarantees. One should be able to go with the tour-de-force analysis analogous to [16, 19]. However, we find it valuable to provide analysis that is tractable using just elementary and short proofs. We show that by taking advantage of the Gumbel

distribution being the limiting distribution, one can isolate a very simple combinatorial problem capturing the essence of the stochastic averaging. The analysis of this problem requires application of only some basic probabilistic inequalities and multinomial identities.¹

2 Related work

The key concept used in virtually all cardinality estimation results, can be summarized as follows: given a universe U of elements, we start by picking a hash-function. Then, given a subset $M \subseteq U$ which cardinality we want to estimate, we proceed by applying h to every element of M and operate only on $M' = \{h(x) : x \in M\} \subset [0, 1]$. The next step is computing an *observable* – i.e. a quantity that only depends on the underlying set and is independent of replications. In the final step we somehow extract the estimate of the cardinality from the observable.

For example [7] uses $h : M \rightarrow [0, 1]$ and the value $y = \min M' = \min_{x \in M} h(x)$ as an observable. We expect $y \sim \frac{1}{n+1}$, thus $\frac{1}{y} - 1$ is used as an estimate of the cardinality n . However, since we need to overcome the variance, one might need to average over many independent instances of the process, in order to achieve a good estimation. In this particular example, to get an $(1 + \varepsilon)$ approximation, we need to average over $\mathcal{O}(\varepsilon^{-2})$ independent copies of the algorithm. Therefore, the total memory usage becomes $\mathcal{O}(\varepsilon^{-2} \log n)$ bits.

Stochastic averaging

Naively averaging over k independent copies of the algorithm has an important drawback - the time for processing each query grows from $O(1)$ to $O(k)$. *Stochastic averaging* is a technique designed to address that issue. In our setting it works as follows: instead of processing each element in each of the k processes independently (which is a bottleneck), we randomly partition our input into k disjoint sub-inputs: $M = M_1 \cup \dots \cup M_k$, and we run each copy of an algorithm only on its corresponding sub-input. This is simulated by picking a second hash function $h' : M \rightarrow \{1, \dots, k\}$, and when we are processing an element x , it is assigned to M_i where $i = h'(x)$ is decided solely on the hash of x . Intuitively, we expect each M_i to contain roughly n/k elements. Note that the actual number of elements in all M_i follows a *multinomial distribution*, and this presents an additional challenge in the analysis.

LogLog sketching

Consider the following: we hash the elements to bitstrings, that is $h : M \rightarrow \{0, 1\}^\infty$, and consider the bit-patterns observed. For each element find the value $\text{bit}(x)$ such that $h(x)$ has a prefix $0^{\text{bit}(x)}1$. The particular value $\text{bit}(x) = c$ should be observed once every $\sim 2^c$ different hashes, and can be used to estimate the cardinality. The observable used in the **LogLog** is the value $\max_x \text{bit}(x)$ among all elements. Since we expect its value to be roughly of order $\log n$, we maintain the value of $\max \text{bit}(x)$ on $\mathcal{O}(\log \log n)$ bits.

Denote the observables produced in the concurrent copies of the algorithm as t_1, \dots, t_k . We expect the values of t_i to be such that $2^{t_i} \sim n/k$. One can easily show, that for any t_i , we have $\mathbb{E}[2^{t_i}] = \infty$, thus taking the arithmetic average over 2^{t_i} is not a feasible strategy.

¹ It is important to note that this is not the **first** cardinality estimation algorithm with a simple analysis, e.g. [7, 20] algorithms have relatively straightforward analysis. However, none of those techniques apply to the simplification of **LogLog** or **HyperLogLog** specifically, which are default practical choices for the cardinality estimation.

However, it turns out that the geometric average works in this setting, and we expect the $k \left(\prod_i 2^{t_i} \right)^{1/k}$ to be an estimate for n (one also needs a normalizing constant that depends solely on k). The analysis in [16] shows that the variance of the estimation is roughly $1.3/\sqrt{k}$.

HyperLogLog sketching

HyperLogLog ([19]) is an improvement over LogLog with the observation that the *harmonic average* achieves a better averaging performance over *geometric average*. Thus HyperLogLog is constructed by setting the estimator to $k^2 \left(\sum_i 2^{-t_i} \right)^{-1}$ with some normalizing constant (depending on k). The resulting algorithm has a variance which is roughly $1.04/\sqrt{k}$.

In fact it can be shown that the harmonic average is optimal in that setting: among observables that constitute of taking maximum of a hash function, harmonic average is a *maximum likelihood estimator* (see e.g. [13]). However, this claim is strict only without stochastic averaging.

Due to the space limitations, in this article we provide only the analysis of the LogLog version (with geometric average estimation) of our algorithm. The HyperLogLog version (using harmonic average estimation) is available in the full version of the paper. ²

3 Preliminaries

Computational model

We assume oracle access to a perfect source of randomness, that is a hash function $h : [u] \rightarrow \{0, 1\}^\infty$. If the sketch demands it, we allow it to access multiple independent such sources, which can be simulated with a help of bit and arithmetic operations. The oracle access is a standard assumption in this line of work (c.f. discussion in [31]) – the purpose of this assumption is to separate the analysis of the space complexity of the algorithm from the space complexity of the source of the randomness.

Besides that, we assume standard RAM model, with words of size u and standard arithmetic operations on those words taking constant time.

Gumbel distribution

We use the following distribution, which originates from the *extreme value theory*.

► **Definition 1** (Gumbel distribution [25]). *Let $\text{Gumbel}(\mu)$ denote the distribution given by a following CDF:*

$$F(x) = e^{-e^{-(x-\mu)}}.$$

Its probability density function is given by

$$f(x) = e^{-e^{-(x-\mu)}} e^{-(x-\mu)}.$$

Observe that, directly from the definition, if $X \sim \text{Gumbel}(\mu)$, then $X+c \sim \text{Gumbel}(\mu+c)$.

We also note that when $x \rightarrow \infty$, then $f(x) \approx e^{-(x-\mu)}$, thus the Gumbel distribution has an exponential tail on the positive side. The distribution has a doubly-exponential tail when $x \rightarrow -\infty$.

² The full version of the paper is available under the following link: <https://arxiv.org/abs/2008.07590>.

76:6 Cardinality Estimation Using Gumbel Distribution

We have the following basic property when $X \sim \text{Gumbel}(\mu)$ (c.f. [25]):

$$\mathbb{E}[e^{\alpha X}] = e^{\alpha\mu} \int_{-\infty}^{\infty} e^{-e^{-x}} e^{(\alpha-1)x} dx = e^{\alpha\mu} \Gamma(1 - \alpha), \quad (1)$$

from which it follows that $\mathbb{E}[e^{-X}] = e^{-\mu}$ and $\text{Var}[e^{-X}] = e^{-2\mu}$.

► **Property 2** (Sampling from Gumbel distribution.). *If $t \in [0, 1]$ is drawn uniformly at random, then $X = -\ln(-\ln t) + \mu$ has the distribution $\text{Gumbel}(\mu)$.*

The following property is the key property used in our algorithm analysis. It essentially states that Gumbel distribution is invariant under taking the maximum of independent samples (up to normalization).³

► **Property 3.** *If $x_1, x_2, \dots, x_n \sim \text{Gumbel}(0)$ are independent random variables, then for $Z = \max(x_1, \dots, x_n)$ we have $Z \sim \text{Gumbel}(\ln n)$.*

Proof.

$$\Pr(Z < x) = \prod_i \Pr(x_i < x) = (e^{-x})^n = e^{-x + \ln n}. \quad \blacktriangleleft$$

Multinomial distribution

We now discuss the multinomial distribution and its role in the analysis of the stochastic averaging.

► **Definition 4.** *We say that X_1, \dots, X_k are distributed according to $\text{Multinomial}(n; p_1, \dots, p_k)$ distribution for some $\sum_i p_i = 1$, if, for any $n_1 + \dots + n_k = n$ there is*

$$\Pr[X_1 = n_1 \wedge \dots \wedge X_k = n_k] = \binom{n}{n_1, \dots, n_k} p_1^{n_1} \dots p_k^{n_k}.$$

Consider a process of distributing n identical balls to k urns, where for each ball we place it in the urn i with probability p_i , fully independently between the balls. Then, the vector of the total number of balls in each urn X_1, \dots, X_k follows $\text{Multinomial}(n; p_1, \dots, p_k)$ distribution.

For our purposes we are interested in the following setting: let f be some real-valued function. Lets say that we have a stochastic process of estimating cardinality in a stream, that is if n distinct elements appear, the process outputs a value that is concentrated around its expected value $f(n)$. Now, we apply stochastic averaging, by splitting the stream into sub-streams, and feed each sub-stream to estimation process separately, say n_i going into sub-stream i . We can look at the following random variables:

$$S_n = \mathbb{E}\left[\sum_i f(n_i)\right] \quad \text{and} \quad P_n = \mathbb{E}\left[\prod_i f(n_i)\right].$$

³ In fact, the Fisher–Tippett–Gnedenko theorem (c.f. [15]) states, that for any distribution \mathcal{D} , if for some a_n, b_n the limit $\lim_{n \rightarrow \infty} \left(\frac{\max(X_1, \dots, X_n) - b_n}{a_n} \right)$ converges to some non-degenerate distribution, where $X_1, \dots, X_n \sim \mathcal{D}$ (and are independent), then it converges to one of three possible distribution families: a Fréchet distribution, a Weibull distribution or a Gumbel distribution. Thus, those three distributions can be viewed as a counterpart to normal distribution, w.r.t. to taking maximum (instead of repeated additions).

We expect $S_n \approx kf(n/k)$ and $P_n \approx f(n/k)^k$. Deriving actual concentration bounds for specifically chosen functions f gives us insight on how well harmonic average or geometric average performs when concentrating cardinality estimation processes under stochastic averaging.

The analysis of the stochastic averaging for a *generic* function f (under some regularity constraints) has been done in [13]. We actually derive a stronger set of bounds for very specific functions: $f(x) = \ln(x+1)$ and $f(x) = \frac{1}{x+1}$.

4 Geometric average estimation

We start by showing a simple concentration result for geometric average of independent random variables distributed according to the Gumbel distribution.

► **Lemma 5.** *Let G_1, \dots, G_k be independent random variables distributed according to $\text{Gumbel}(0)$, and let $G = \sum_i G_i$. If $k > 1$, then $\mathbb{E}[\exp(G/k)] = \Gamma(1 - 1/k)^k = \exp(\gamma)(1 + \frac{\pi^2}{12} \frac{1}{k} + \mathcal{O}(k^{-2}))$. If $k > 2$, then $\text{Var}[\exp(G/k)] = \Gamma(1 - 2/k)^k - \Gamma(1 - 1/k)^{2k} = \exp(2\gamma) \cdot (\frac{\pi^2}{6k} + \mathcal{O}(k^{-2}))$*

Proof.

$$\begin{aligned} \mathbb{E}[\exp(G/k)] &= \prod_i \mathbb{E}[\exp(G_i/k)] = \Gamma(1 - 1/k)^k \\ \text{Var}[\exp(G/k)] &= \prod_i \mathbb{E}[\exp(G_i/k)^2] - \prod_i \mathbb{E}[\exp(G_i/k)]^2 \\ &= \Gamma(1 - 2/k)^k - \Gamma(1 - 1/k)^{2k}. \end{aligned}$$

From the Taylor expansion of the log-gamma function we get that $\Gamma(1 - z) = \exp(\gamma z + \frac{\pi^2}{12} z^2 + \mathcal{O}(z^3))$, which yields the desired approximations. ◀

The following algorithm shows that if we are fine with slower updates, then the Gumbel distribution fits nicely into the standard cardinality estimation framework. The main idea is just to hash each element into a real-value distributed according to Gumbel distribution, and take the maximum across the values.

■ **Algorithm 1** Cardinality estimation using Gumbel distribution.

```

1 Procedure INIT()
2   pick  $r_1, \dots, r_k : U \rightarrow [0, 1]$  as independent hash functions
3    $X_1 \leftarrow -\infty, \dots, X_k \leftarrow -\infty$ 
4 Procedure UPDATE( $x$ )
5   for  $1 \leq i \leq k$  do
6      $v \leftarrow -\ln(-\ln r_i(x))$  //  $\text{Gumbel}(0)$  RV
7      $X_i \leftarrow \max(v, X_i)$ 
8 Procedure GEOMETRICESTIMATE()
9    $\alpha_k \leftarrow \Gamma(1 - 1/k)^{-k}$  // normalizing factor, for large  $k$ :  $\alpha_k \approx \exp(-\gamma)$ 
10  return  $Z = \exp(\frac{1}{k} \sum_i X_i) \cdot \alpha_k$ 

```

► **Theorem 6.** *Applied to a stream of n distinct elements, Algorithm 1 outputs Z such that $\mathbb{E}[Z] = n$ and if $k > 2$ then $\text{Var}[Z] = \frac{n^2}{k} \left(\frac{\pi^2}{6} + O(k^{-1}) \right)$. It uses k real-value registers and spends $\mathcal{O}(k)$ operations per single processed element of the input.*

Proof. We analyze the Algorithm 1 after processing a stream of n distinct elements. For each X_i , its value is a maximum of n random variables drawn from Gumbel(0) distribution, so by Property 3 we have that $X_i \sim \text{Gumbel}(\ln n)$. Hence, $X_i = G_i + \ln n$ where all G_i are identically distributed according to the Gumbel(0). Moreover, repeated occurrences of the elements do not change the state of the algorithm. Denoting $G = \sum_i G_i$, we get

$$\mathbb{E}[Z] = \Gamma(1 - 1/k)^{-k} \mathbb{E}[\exp(\ln n + G/k)] = n\Gamma(1 - 1/k)^{-k}\Gamma(1 - 1/k)^k = n$$

and

$$\text{Var}[Z] = n^2\Gamma(1 - 1/k)^{-2k} \text{Var}[\exp(G/k)] = n^2 \left(\frac{\Gamma(1 - 2/k)^k}{\Gamma(1 - 1/k)^{2k}} - 1 \right). \quad \blacktriangleleft$$

4.1 Stochastic averaging

We refine the Algorithm 1 by adding stochastic averaging. Application of the technique is straightforward, but for technical reasons we need to take care of the initialization of the registers – since the expected value of X_i is the logarithm of the number of the elements assigned to the i -th register, we don't want any of the registers to be empty at the and. Therefore, at the beginning we feed each of them with an artificial random element.

■ **Algorithm 2** Cardinality estimation using Gumbel distribution and stochastic averaging.

```

1 Procedure INIT()
2   pick  $h : U \rightarrow \{1, \dots, k\}$  and  $r : U \rightarrow [0, 1]$  as independent hash functions
3   for  $1 \leq i \leq m$  do
4      $X_i \leftarrow -\ln(-\ln u_i)$  where  $u_i$  is picked uniformly from  $[0, 1]$ . // Gumbel(0) RV
5 Procedure UPDATE( $x$ )
6    $t \leftarrow h(x)$ 
7    $v \leftarrow -\ln(-\ln r(x))$  // Gumbel(0) RV
8    $X_t \leftarrow \max(v, X_t)$ 
9 Procedure GEOMETRICESTIMATE()
10   $\alpha_k \leftarrow \Gamma(1 - 1/k)^{-k}$  // normalizing factor, for large  $k$ :  $\alpha_k \approx \exp(-\gamma)$ 
11  return  $Z = k \cdot \exp(\frac{1}{k} \sum_i X_i) \cdot \alpha_k$ 

```

► **Theorem 7.** *Applied to a stream of n distinct elements, Algorithm 2 outputs Z such that if $k > 1$ then $n \frac{k}{k+1} \leq \mathbb{E}[Z] \leq n + k$ and if $k > 2$ then $\text{Var}[Z] \leq 3.645 \frac{n^2}{k} + \mathcal{O}(n^2/k^2 + k^2)$. It uses k real-value registers and spends constant number of operations per single processed element of the input.*

Proof. We analyze Algorithm 2 after processing stream S of n distinct elements. Let n_1, \dots, n_k be the respective numbers of unique items hashed by h into registers $\{1, \dots, k\}$ respectively. It follows that $n_1, \dots, n_k \sim \text{Multinomial}(n; \frac{1}{k}, \dots, \frac{1}{k})$. For each X_i , its value is a maximum of $n_i + 1$ random variables drawn from the Gumbel(0) distribution (taking into account n_i updates to its value and the initialization). Thus, conditioned on the specific

values of n_1, \dots, n_k , we have that X_i follows the Gumbel distribution – more specifically $X_i | n_1, \dots, n_k \sim \text{Gumbel}(\ln(n_i + 1))$. Let us denote $G_i = X_i - \ln(n_i + 1)$, $G = \sum_i G_i$ and $Y = \sum_i \ln(n_i + 1)$. We observe that G_i 's are independent random variables distributed according to $\text{Gumbel}(0)$ (and independent from Y).

We now have

$$Z = k\Gamma(1 - 1/k)^{-k} \exp(Y/k) \exp(G/k).$$

Since $\mathbb{E}[\exp(G/k)] = \Gamma(1 - 1/k)^k$ and G and Y are independent, using Lemma 8 we get

$$\mathbb{E}[Z] = k \mathbb{E}[\exp(Y/k)] \geq n \cdot \frac{k}{k+1}$$

and

$$\mathbb{E}[Z] = k \mathbb{E}[\exp(Y/k)] \leq n + k.$$

Now, using $\text{Var}[AB] = \text{Var}[A] \mathbb{E}[B^2] + \mathbb{E}[A]^2 \text{Var}[B]$ identity for independent random variables and Lemma 8 we can bound

$$\begin{aligned} \text{Var}[Z] &= k^2 \Gamma(1 - 1/k)^{-2k} (\text{Var}[\exp(Y/k)] \mathbb{E}[\exp(G/k)^2] + \mathbb{E}[\exp(Y/k)]^2 \text{Var}[\exp(G/k)]) \\ &\leq (k^2 + 2n^2/k + O(n^2/k^2)) \frac{\Gamma(1 - 2/k)^k}{\Gamma(1 - 1/k)^{2k}} + (n + k)^2 \left(\frac{\Gamma(1 - 2/k)^k}{\Gamma(1 - 1/k)^{2k}} - 1 \right) \\ &= (k^2 + 2n^2/k + O(n^2/k^2))(1 + O(k^{-1})) + (n + k)^2 \left(\frac{\pi^2}{6k} + O(k^{-2}) \right), \end{aligned}$$

and the claim follows. ◀

► **Lemma 8.** *Let $n_1, \dots, n_k \sim \text{Multinomial}(n; 1/k, \dots, 1/k)$ and let $T = \sqrt[k]{\prod_i (n_i + 1)} = \exp(\frac{1}{k} \sum_i \ln(n_i + 1))$. Then there is $n/(k+1) \leq \mathbb{E}[T] \leq n/k + 1$ and $\text{Var}[T] \leq 1 + 2n^2/k^3 + \mathcal{O}(n^2/k^4)$.*

Proof. Denote $Y = \sum_i \ln(n_i + 1)$. By Lemma 9 bound we have

$$\begin{aligned} \mathbb{E}[\exp(Y/k)] &\geq \int_0^\infty \exp(\ln(n/k) - t/k) e^{-t} dt \\ &= n/k \int_0^\infty \exp(-\frac{k+1}{k}t) dt \\ &= n/(k+1). \end{aligned}$$

By concavity of a logarithm we have $Y = \sum_i \ln(n_i + 1) \leq k \ln(n/k + 1)$, so $\exp(Y/k) \leq n/k + 1$. Finally, by Lemma 9 bound we get

$$\begin{aligned} \text{Var}[\exp(Y/k)] &\leq \mathbb{E}[(\exp(Y/k) - n/k)^2] \\ &\leq ((n/k + 1) - n/k)^2 + \frac{n^2}{k^2} \int_0^\infty (1 - e^{-t/k})^2 e^{-t} dt \\ &= 1 + \frac{n^2}{k^2} \left(1 - 2\frac{k}{k+1} + \frac{k}{k+2} \right). \end{aligned} \quad \blacktriangleleft$$

► **Lemma 9.** *Let $n_1, \dots, n_k \sim \text{Multinomial}(n; 1/k, \dots, 1/k)$ and let $Y = \sum_i \ln(n_i + 1)$. Then $Y \geq k \ln(n/k) - t$ with probability at least $1 - e^{-t}$.*

76:10 Cardinality Estimation Using Gumbel Distribution

Proof. Consider $\mathbb{E}[e^{-Y}]$. We have

$$\begin{aligned}
 \mathbb{E}_{\text{Multinomial}}^{n_1, \dots, n_k \sim} [e^{-Y}] &= \mathbb{E}_{\text{Multinomial}}^{n_1, \dots, n_k \sim} \left[\prod_i \frac{1}{n_i + 1} \right] \\
 &= \sum_{i_1 + \dots + i_k = n} \Pr[n_1 = i_1 \wedge \dots \wedge n_k = i_k] \prod_i \frac{1}{i_i + 1} \\
 &= \sum_{i_1 + \dots + i_k = n} k^{-n} \binom{n}{i_1, \dots, i_k} \prod_i \frac{1}{i_i + 1} \\
 &= k^{-n} \sum_{i_1 + \dots + i_k = n} \frac{n!}{(i_1 + 1)! \dots (i_k + 1)!} \\
 &= k^{-n} \sum_{i_1 + \dots + i_k = n} \binom{n+k}{i_1 + 1, \dots, i_k + 1} \frac{n!}{(n+k)!} \\
 &\leq k^{-n} k^{n+k} \frac{n!}{(n+k)!} \\
 &\leq \left(\frac{k}{n}\right)^k.
 \end{aligned}$$

Thus, for any $t > 0$, by Markov's inequality

$$\begin{aligned}
 \Pr[Y \leq k \ln(n/k) - t] &= \Pr[e^{-Y} \geq e^{t - k \ln(n/k)}] \\
 &\leq \Pr[e^{-Y} \geq e^t \cdot \mathbb{E}[e^{-Y}]] \\
 &\leq e^{-t}.
 \end{aligned}$$

4.2 Discretization

Presented sketches use k real-value registers, which is in disadvantage when compared with LogLog and HyperLogLog, where only k integers are used, each taking $\mathcal{O}(\log \log n)$ bits. We now discuss how to reduce the memory footprint of the algorithms. This section exemplifies the usefulness of Gumbel distributions. In particular, this is a family of the limit distributions where *additive error* of registers corresponds to *multiplicative error* of estimation.

Simple rounding

First, we note that rounding the registers to nearest multiplicity of ε for some $\varepsilon > 0$ introduces at most $\exp(\varepsilon) = 1 + \varepsilon + \mathcal{O}(\varepsilon^2)$ multiplicative distortion in the estimation procedure `GeometricEstimate()` from both Algorithm 1 and 2. For example, for 1, we have, assuming X'_i are rounded registers: $|X'_i - X_i| \leq \varepsilon$, and so for $Z' = \alpha_k \exp(\frac{1}{k} \sum_i X'_i)$ there is $\frac{Z'}{Z} = \exp(\frac{1}{k} \sum_i (X'_i - X_i))$, so $\exp(-\varepsilon) \leq \frac{Z'}{Z} \leq \exp(\varepsilon)$. Since each register stores w.h.p. values of magnitude $2 \log n$, it can be implemented on integer registers using $\mathcal{O}(\log \frac{\log n}{\varepsilon}) = \mathcal{O}(\log \log n + \log \varepsilon^{-1})$ bits.

Randomized rounding

We now show how to eliminate the $\log \varepsilon^{-1}$ term. We define the following *shift-rounding*, for shift value $c \in [0, 1)$:

$$f_c(x) \stackrel{\text{def}}{=} \lfloor x + c \rfloor - c.$$

We note two key properties:

1. shift-rounding commutes with maximum, that is, for any x_1, \dots, x_k , we have $\max(f_c(x_1), \dots, f_c(x_k)) = f_c(\max(x_1, \dots, x_k))$,
2. If $c \sim U[0, 1]$, then $f_c(x) \sim U[x - 1, x]$, where $U[a, b]$ denotes uniform distribution on range $[a, b]$.

We thus show how to adapt the Algorithm 2 using shift-rounding.

The analysis of Algorithm 3 comes from following invariant: if Algorithms 3 and 2 are run side-by-side on the same input stream, at any given moment there is $X'_i = f_{c_i}(X_i)$. Thus, we have the following $X'_i \sim \text{Gumbel}(\ln(n_i + 1)) - U[0, 1] = \ln(n_i + 1) + \text{Gumbel}(0) - U[0, 1]$.

■ **Algorithm 3** Algorithm 2 with shift-rounding.

```

1 Procedure INIT()
2   pick  $h : U \rightarrow \{1, \dots, k\}$  and  $r : U \rightarrow [0, 1]$  as independent hash functions
3   for  $1 \leq i \leq m$  do
4      $c_i$  is picked uniformly from  $[0, 1]$ 
5      $X'_i \leftarrow \lfloor -\ln(-\ln u_i) + c_i \rfloor - c_i$ 
6     where  $u_i$  is picked uniformly from  $[0, 1]$ . // Gumbel(0) RV + randomized
7     rounding
7 Procedure UPDATE( $x$ )
8    $t \leftarrow h(x)$ 
9    $v \leftarrow \lfloor -\ln(-\ln h(x)) + u_i \rfloor - u_i$ 
10   $X'_i \leftarrow \max(v, X'_i)$ 
11 Procedure GEOMETRICESTIMATE()
12   $\alpha'_k \leftarrow \Gamma(1 - 1/k)^{-k} (1 - \exp(-1/k))^{-k} k^{-k}$  // normalizing factor, for large
13   $k: \alpha'_k \approx \exp(1/2 - \gamma)$ 
14  return  $Z = k \cdot \exp(\frac{1}{k} \sum_i X'_i) \cdot \alpha'_k$ 

```

Additionally, X'_i are independent as X_i were independent. We observe that for $X' \sim \text{Gumbel}(0) - U[0, 1]$, there is $\mathbb{E}[\exp(X'/k)] = \Gamma(1 - 1/k)(1 - \exp(-1/k))k$, so we have equivalents of Lemma 5 in the following sense: $\mathbb{E}[\exp(\frac{1}{k} \sum_i G'_i)] = \Gamma(1 - 1/k)^k (1 - \exp(-1/k))^k k^k \approx \exp(\gamma - 1/2)(1 + (\frac{\pi^2}{12} + \frac{1}{6})\frac{1}{k} + \mathcal{O}(k^{-2}))$, and $\text{Var}[\exp(\frac{1}{k} \sum_i G'_i)] \approx \exp(2\gamma - 1)((\frac{\pi^2}{6} + \frac{1}{3})\frac{1}{k} + \mathcal{O}(k^{-2}))$.

Thus an equivalent of Theorem 7 applies to Algorithm 3 with slightly worse constants.

► **Theorem 10.** *Applied to a stream of n distinct elements, Algorithm 3 outputs Z such that if $k > 1$ then $n \frac{k}{k+1} \leq \mathbb{E}[Z] \leq n + k$ and if $k > 2$ then $\text{Var}[Z] \leq 3.98 \frac{n^2}{k} + \mathcal{O}(n^2/k^2 + k^2)$. It uses k integer registers of size $\mathcal{O}(\log \log n)$ bits each and spends constant number of operations per single processed element of the input.*

We note that each X'_i takes values only from set $\mathbb{Z} - c_i$ of magnitude at most $2 \log n$, it can be stored using $\mathcal{O}(\log \log n)$ bits. Values of c_i do not need to be stored explicitly, as those can be extracted by picking a hash function $c : \{1, \dots, k\} \rightarrow [0, 1]$ and setting $c_i = c(i)$.

References

- 1 Algebird HyperLogLog implementation. Accessed: 2022-04-21. URL: <https://twitter.github.io/algebird/datatypes/approx/hyperloglog.html>.
- 2 Counting uniques faster in BigQuery with HyperLogLog++. Accessed: 2022-04-21, URL: <https://cloud.google.com/blog/products/gcp/counting-uniques-faster-in-bigquery-with-hyperloglog>.
- 3 HyperLogLog Sketch. Accessed: 2022-04-21. URL: <https://datasketches.apache.org/docs/HLL/HLL.html>.
- 4 Presto HyperLogLog function. Accessed: 2022-04-21. URL: <https://prestodb.github.io/docs/current/functions/hyperloglog.html>.
- 5 Redis PFCOUNT command. Accessed: 2022-04-21. URL: <https://redis.io/commands/pfcount>.
- 6 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996. doi:10.1145/237814.237823.
- 7 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *RANDOM 2002*, pages 1–10, 2002. doi:10.1007/3-540-45726-7_1.
- 8 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA 2002*, pages 623–632. ACM/SIAM, 2002. URL: <http://dl.acm.org/citation.cfm?id=545381.545464>.
- 9 Kevin Beyer, Rainer Gemulla, Peter J Haas, Berthold Reinwald, and Yannis Sismanis. Distinct-value synopses for multiset operations. *Communications of the ACM*, 52(10):87–95, 2009.
- 10 Joshua Brody and Amit Chakrabarti. A multi-round communication lower bound for gap hamming and some consequences. In *CCC 2009*, pages 358–368, 2009. doi:10.1109/CCC.2009.31.
- 11 Jarosław Blasiok. Optimal streaming and tracking distinct elements with high probability. In *SODA 2018*, pages 2432–2448, 2018. doi:10.1137/1.9781611975031.156.
- 12 Aiyou Chen, Jin Cao, Larry Shepp, and Tuan Nguyen. Distinct counting with a self-learning bitmap. *Journal of the American Statistical Association*, 106(495):879–890, 2011.
- 13 Peter Clifford and Ioana A Cosma. A statistical analysis of probabilistic counting algorithms. *Scandinavian Journal of Statistics*, 39(1):1–14, 2012.
- 14 Edith Cohen. All-distances sketches, revisited: Hip estimators for massive graphs analysis. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2320–2334, 2015.
- 15 Laurens De Haan and Ana Ferreira. *Extreme value theory: an introduction*. Springer Science & Business Media, 2007.
- 16 Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In *ESA 2003*, pages 605–617, 2003. doi:10.1007/978-3-540-39658-1_55.
- 17 Otmar Ertl. New cardinality estimation algorithms for hyperloglog sketches. *CoRR*, abs/1702.01284, 2017. arXiv:1702.01284.
- 18 Cristian Estan, George Varghese, and Michael E. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, 2006. doi:10.1145/1217709.
- 19 Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.
- 20 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985. doi:10.1016/0022-0000(85)90041-8.
- 21 Lucas Gerin and Philippe Chassaing. Efficient estimation of the cardinality of large data sets. *Discrete Mathematics & Theoretical Computer Science*, 2006.

- 22 Phillip B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB 2001*, pages 541–550, 2001. URL: <http://www.vldb.org/conf/2001/P541.pdf>.
- 23 Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *SPAA 2001*, pages 281–291, 2001. doi:10.1145/378580.378687.
- 24 Frédéric Giroire. Order statistics and estimating cardinalities of massive data sets. *Discret. Appl. Math.*, 157(2):406–427, 2009. doi:10.1016/j.dam.2008.06.020.
- 25 Emil Julius Gumbel. Les valeurs extrêmes des distributions statistiques. In *Annales de l'Institut Henri Poincaré*, volume 5(2), pages 115–158, 1935.
- 26 Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. In *EDBT 2013*, pages 683–692, 2013. doi:10.1145/2452376.2452456.
- 27 Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem. In *FOCS 2003*, pages 283–288, 2003. doi:10.1109/SFCS.2003.1238202.
- 28 T. S. Jayram and David P. Woodruff. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with sub-constant error. In *SODA 2011*, pages 1–10, 2011. doi:10.1137/1.9781611973082.1.
- 29 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *PODS 2010*, pages 41–52, 2010. doi:10.1145/1807085.1807094.
- 30 Jérémie Lumbroso. An optimal cardinality estimation algorithm based on order statistics and its full analysis. *Discrete Mathematics & Theoretical Computer Science*, 2010.
- 31 Seth Pettie and Dingyu Wang. Information theoretic limits of cardinality estimation: Fisher meets shannon. In *STOC 2021*, pages 556–569. ACM, 2021.
- 32 Seth Pettie, Dingyu Wang, and Longhui Yin. Non-mergeable sketching for cardinality estimation. In *ICALP 2021*, volume 198 of *LIPICs*, pages 104:1–104:20, 2021.
- 33 Wojciech Szpankowski. *Average case analysis of algorithms on sequences*, volume 50. John Wiley & Sons, 2011.
- 34 Daniel Ting. Streamed approximate counting of distinct elements: beating optimal batch methods. In *KDD 2014*, pages 442–451. ACM, 2014. doi:10.1145/2623330.2623669.
- 35 Alfredo Viola, Conrado Martínez, Jérémie Lumbroso, and Ahmed Helmi. Data streams as random permutations: the distinct element problem. *Discrete Mathematics & Theoretical Computer Science*, 2012.
- 36 David P. Woodruff. Optimal space lower bounds for all frequency moments. In *SODA 2004*, pages 167–175, 2004. URL: <http://dl.acm.org/citation.cfm?id=982792.982817>.
- 37 Qingjun Xiao, You Zhou, and Shigang Chen. Better with fewer bits: Improving the performance of cardinality estimation of large data streams. In *INFOCOM 2017*, pages 1–9, 2017.

(In-)Approximability Results for Interval, Resource Restricted, and Low Rank Scheduling

Marten Maack ✉ 

Heinz Nixdorf Institute & Department of Computer Science, Universität Paderborn, Germany

Simon Pukrop ✉ 

Heinz Nixdorf Institute & Department of Computer Science, Universität Paderborn, Germany

Anna Rodriguez Rasmussen ✉

Department of Mathematics, Uppsala University, Sweden

Abstract

We consider variants of the restricted assignment problem where a set of jobs has to be assigned to a set of machines, for each job a size and a set of eligible machines is given, and the jobs may only be assigned to eligible machines with the goal of makespan minimization. For the variant with interval restrictions, where the machines can be arranged on a path such that each job is eligible on a subpath, we present the first better than 2-approximation and an improved inapproximability result. In particular, we give a $(2 - \frac{1}{24})$ -approximation and show that no better than $9/8$ -approximation is possible, unless $P=NP$. Furthermore, we consider restricted assignment with R resource restrictions and rank D unrelated scheduling. In the former problem, a machine may process a job if it can meet its resource requirements regarding R (renewable) resources. In the latter, the size of a job is dependent on the machine it is assigned to and the corresponding processing time matrix has rank at most D . The problem with interval restrictions includes the 1 resource variant, is encompassed by the 2 resource variant, and regarding approximation the R resource variant is essentially a special case of the rank $R + 1$ problem. We show that no better than $3/2$, $8/7$, and $3/2$ -approximation is possible (unless $P=NP$) for the 3 resource, 2 resource, and rank 3 variant, respectively. Both the approximation result for the interval case and the inapproximability result for the rank 3 variant are solutions to open challenges stated in previous works. Lastly, we also consider the reverse objective, that is, maximizing the minimal load any machine receives, and achieve similar results.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling, Restricted Assignment, Approximation, Inapproximability

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.77

Related Version *Full Version:* <https://arxiv.org/abs/2203.06171> [13]

Funding This work was supported by the German Research Foundation (DFG) within the Collaborative Research Centre “On-The-Fly Computing” under the project number 160364472 – SFB 901/3.

1 Introduction

Makespan minimization on unrelated parallel machines, or unrelated scheduling for short, is considered a fundamental problem in approximation and scheduling theory. In this problem, a set \mathcal{J} of jobs has to be assigned to a set \mathcal{M} of machines via a schedule $\sigma : \mathcal{J} \rightarrow \mathcal{M}$. Each job j has a processing time p_{ij} depending on the machine i it is assigned to and the goal is to minimize the makespan $C_{\max}(\sigma) = \max_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_{ij}$. In 1990, Lenstra, Shmoys, and Tardos [10] presented a 2-approximation for this problem and further showed that no better than 1.5-approximation can be achieved (unless $P=NP$) already for the restricted assignment problem, where each job j has a size p_j and $p_{ij} \in \{p_j, \infty\}$ for each machine i .



© Marten Maack, Simon Pukrop, and Anna Rodriguez Rasmussen; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 77; pp. 77:1–77:13

Leibniz International Proceedings in Informatics



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

For each job j we denote its set of eligible machines by $\mathcal{M}(j) = \{i \in \mathcal{M} \mid p_{ij} = p_j\}$. Closing or narrowing the gap between 2-approximation and 1.5-inapproximability is a famous open problem in approximation [18] and scheduling theory [15]. The present paper deals with certain subproblems of both unrelated scheduling and restricted assignment.

Interval Restrictions. In the variant of restricted assignment with interval restrictions, denoted as RAI in the following, there is a total order of the machines and each job j is eligible on a discrete interval of machines, i.e., $\mathcal{M} = \{M_1, M_2, \dots, M_m\}$ and $\mathcal{M}(j) = \{M_\ell, M_{\ell+1}, \dots, M_r\}$ for some $\ell, r \in [m]$. There are several variants and special cases of this problem that are known to admit a polynomial time approximation scheme (PTAS), see [6, 8, 9, 11, 14, 16], the most prominent of which is probably the hierarchical case [11] in which each job is eligible on an interval of the form $\{M_1, M_2, \dots, M_r\}$, i.e., the first machine is eligible for each job. For RAI, on the other hand, there is an $(1 + \delta)$ -inapproximability result for some small but constant $\delta > 0$ [12]. Furthermore, Schwarz [16] designed a $(2 - 2/(\max_{j \in \mathcal{J}} p_j))$ -approximation (assuming integral processing times); and Wang and Sitters [17] studied an LP formulation that provides an optimal solution for the special case with two distinct processing times and some additional assumption.

Resource Restrictions. In the restricted assignment problem with R resource restrictions, or $\text{RAR}(R)$, a set \mathcal{R} of R (renewable) resources is given, each machine i has a resource capacity $c_r(i)$ and each job j has a resource demand $d_r(j)$ for each $r \in \mathcal{R}$. The eligible machines are determined by the corresponding resource constraints, i.e., $\mathcal{M}(j) = \{i \in \mathcal{M} \mid \forall r \in \mathcal{R} : d_r(j) \leq c_r(i)\}$ for each job j . It is easy to see, that $\text{RAR}(1)$ corresponds to the mentioned hierarchical case which admits a PTAS [11]. On the other hand, there can be no approximation algorithm with ratios smaller than $48/47 \approx 1.02$ or 1.5 for $\text{RAR}(2)$ and $\text{RAR}(4)$, respectively, unless $\text{P}=\text{NP}$, see [12]. The same paper also points out that the case with one resource is a special case of the interval case which in turn is a special case of the two resource case, i.e., $\text{RAR}(1) \subset \text{RAI} \subset \text{RAR}(2)$. While the hierarchical case, i.e. $\text{RAR}(1)$, has been studied extensively before, $\text{RAR}(R)$ was first introduced in a work by Bhaskara et al. [1], who mentioned it as a special case of the next problem that we consider.

Low Rank Scheduling. In the rank D version of unrelated scheduling, or $\text{LRS}(D)$, the processing time matrix (p_{ij}) has a rank of at most D . Alternatively (see [3]), we can assume that each job j has a D dimensional size vector $s(j)$ and each machine i a D dimensional speed vector $v(i)$ such that $p_{ij} = \sum_{k=1}^D s_k(j)v_k(i)$. Now, $\text{LRS}(1)$ is exactly makespan minimization on uniformly related parallel machines, which is well known to admit a PTAS [7]. Bhaskara et al. [1], who introduced $\text{LRS}(D)$, presented a QPTAS for $\text{LRS}(2)$ along with some initial inapproximability results for $D > 2$. Subsequently, Chen et al. [4] showed that there can be no better than 1.5-approximation for $\text{LRS}(4)$ unless $\text{P}=\text{NP}$, and for $\text{LRS}(3)$ the same authors together with Marx [3] ruled out a PTAS. On an intuitive level, resource restrictions can be seen as a restricted assignment version of low rank scheduling. However, there is a more direct relationship between the two problems: for each $\text{RAR}(R)$ instance there exist $\text{LRS}(R + 1)$ instances that are arbitrarily good approximations of the former (see [12]). Hence, any approximation algorithm for $\text{LRS}(R + 1)$ can also be used for $\text{RAR}(R)$, and any inapproximability result for $\text{RAR}(R)$ carries over to $\text{LRS}(R + 1)$. In fact many (but not all) inapproximability results for low rank scheduling essentially have this form.

Results. We present improved approximation and inapproximability results for this family of problems. In particular:

- An approximation algorithm for RAI with ratio $2 - \frac{1}{24} \approx 1.96$ presented in Section 2;
- a reduction that rules out a better than 1.5 approximation unless $P=NP$, i.e., a 1.5-inapproximability result, for RAR(3) presented in Section 3.1;
- a $8/7$ -inapproximability result for RAR(2) not included in this version of the paper;
- a $9/8$ -inapproximability result for RAI presented in Section 3.2;
- and a 1.5-inapproximability result for LRS(3) not included in this version of the paper.

All the missing proofs and results can be found in the long version of the paper. The positive result for RAI can be considered the first of the two main contributions of this paper. Finding a better than 2-approximation for RAI was posed as an open challenge in previous works [8, 16, 17]. When considering the respective results in [17] and [16], in particular, it seems highly probable that the actual goal of the research was to address exactly that challenge. The presented approximation algorithm follows the approach of solving and rounding a relaxed linear programming formulation of the problem, which has been used in the classical work by Lenstra et al. [10] and many of the results thereafter. In particular, we extend the so called assignment LP due to Lenstra et al. [10] and design a customized rounding approach. Both the linear programming extension and the rounding approach utilize extensions and refinements of ideas from [16] and [17]. Our result joins the relatively short list of special cases of the restricted assignment problem that do not allow a PTAS and for which an approximation algorithm with rate smaller than 2 is known. Other notable entries are the restricted assignment problem with only two processing times [2] and the so-called graph balancing case [5], where each job is eligible on at most two machines.

The inapproximability results directly build upon the results presented in the paper [12], which in turn utilizes many of the previously published ideas, e.g., from [1, 3–5, 10]. We use the satisfiability problem presented in [12] as the starting point for all of our reductions. For the RAI result in particular, we refine and restructure the respective results from [12] aiming for a significantly better ratio. The respective reduction involves a sorting process and curiously the main improvement in the reduction involves changing a sorting process resembling insertion sort into one resembling bubble sort. Due to this change, the construction becomes locally less complex enabling the use of smaller processing times and hence a stronger inapproximability result. Furthermore, the simplified construction in the result enables us to use the basic structure of the reduction as a starting point for the second main result of the paper, namely, the 1.5-inapproximability result for RAR(3). For this reduction several additional considerations and gadgets are needed, arguably making it the most elaborate of the presented results. The search for an inapproximability result with a reasonably big ratio for RAR(3) was stated as an open challenge in the long version of [3]. Adding the new result yields a very clear picture regarding the approximability of low rank makespan minimization: There is a PTAS for LRS(1), a QPTAS for LRS(2), and a 1.5-inapproximability result for LRS(D) with $D \geq 3$. The last two reductions regarding RAR(2) and RAR(3) yield much improved inapproximability results for the respective problems using comparatively simple and elegant reductions. The result regarding RAR(3), in particular, closes a gap in the results of [12] and also yields an (arguably) easier, alternative proof for the result of [4]. Finally, we note that all of the inapproximability results regarding restricted assignment with resource restrictions can be directly applied to the so called fair allocation or santa claus versions of the problems. In these problem variants, we maximize the minimum load received by the machines rather than minimization of the maximum load, i.e., the objective function is given by $C_{\min}(\sigma) = \min_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_{ij}$ in this case.

Further Related Work. For a more detailed discussion of related work, we refer to [12] and the long version of this paper.

2 Approximation Algorithm for Makespan Minimization with Interval Restrictions

We establish the first approximation for RAI with an approximation factor better than 2:

► **Theorem 1.** *There is a $(2 - \gamma)$ -approximation for RAI with $\gamma = \frac{1}{24}$.*

The particular value of the parameter γ is justified in the end. To achieve this result, we first formulate a customized linear program based on the assignment LP due to Lenstra et al. [10] and develop a rounding approach that places different types of jobs in phases. Note that the placement of big jobs with size close to OPT (where OPT is the makespan of an optimal schedule) is often critical when aiming for an approximation ratio of smaller than 2 for a makespan minimization problem. For instance, the classical 2-approximation [10] for restricted assignment produces a schedule of length at most $\text{OPT} + \max_{j \in \mathcal{J}} p_j$ where OPT is the makespan of an optimal schedule and hence the approximation ratio is better if $\max_{j \in \mathcal{J}} p_j$ is strictly smaller than OPT. This is also the case with our approach – the main effort goes into the careful placement of such big jobs. In particular, we place the largest jobs in a first rounding step and the remaining big jobs in a second. All of these jobs have the property that each machine should receive at most one of them and they are placed accordingly. Moreover, the placement is designed to deviate not too much from the fractional placement due to the LP solution. In a last step, the remaining jobs are placed. Each rounding step is based on a simple heuristic approach that considers the machines from left to right and places the least flexible eligible jobs first, i.e., the jobs that have not been placed yet, are eligible on the current machine, and have a minimal last eligible machine in the ordering of the machines. Both the LP and the rounding approach reuse ideas from [16, 17]. Hence, the main novelty lies in the much more elaborate approach for placing the mentioned big jobs in two phases.

In the following, we first establish some preliminary considerations; then formulate the LP and argue that it is indeed a relaxation of the problem at hand; and then discuss and analyze the different phases of the rounding procedure step by step.

Preliminaries. For any integer k , we set $[k] = \{0, \dots, k - 1\}$. We apply the standard technique (see [10]) of using a binary search framework to guess a candidate makespan T . The goal is then to either correctly decide that no schedule with makespan T exists, or to produce a schedule with makespan at most $(2 - \gamma)T$. Given this guess T , we divide the jobs j into small ($p_j \leq 0.5T$), large ($0.5T < p_j \leq (0.5 + \xi)T$) and huge ($(0.5 + \xi)T < p_j$) jobs depending on some parameter $\xi = \frac{1}{24}$ which is justified later on. We denote the sets of small, large, and huge jobs as \mathcal{S} , \mathcal{L} , and \mathcal{H} , respectively. Furthermore, we fix the (total) order of the machines such that each job is eligible on consecutive machines. This is possible since we are considering RAI. For the sake of simplicity, we assume $\mathcal{M} = [m]$ with the ordering corresponding to the natural one and set $\mathcal{M}(\ell, r) = \{\ell, \dots, r\}$ for each $\ell, r \in \mathcal{M}$. When considering the machines, we use a left to right intuition with predecessor machines on the left and successor machines on the right. Note, that for each job j there exists a left-most and right-most eligible machine and we denote these by $\ell(j)$ and $r(j)$, respectively, i.e., $\mathcal{M}(j) = \mathcal{M}(\ell(j), r(j))$. For a set of jobs $J \subseteq \mathcal{J}$, we call a job $j \in J$ *least flexible* in J if $r(j)$ is minimal in $\{r(j') \mid j' \in J\}$, and a job j is called *less flexible* than a job j' if $r(j) \leq r(j')$. Lastly, we set $J(\ell, r) = \{j \in \mathcal{J} \mid \mathcal{M}(j) \subseteq \mathcal{M}(\ell, r)\}$ for each set of jobs $J \subseteq \mathcal{J}$ and pair of machines $\ell, r \in \mathcal{M}$, and $p(J) = \sum_{j \in \mathcal{J}} p_j$.

Linear Program. The classical assignment LP (see [10]) is given by assignment variables $x_{ij} \in [0, 1]$ for each $i \in \mathcal{M}$ and $j \in \mathcal{J}$ and the following constraints:

$$\sum_{i \in \mathcal{M}} x_{ij} = 1 \quad \forall j \in \mathcal{J} \quad (1)$$

$$\sum_{j \in \mathcal{J}} p_j x_{ij} \leq T \quad \forall i \in \mathcal{M} \quad (2)$$

$$x_{ij} = 0 \quad \forall j \in \mathcal{J}, i \in \mathcal{M} \setminus \mathcal{M}(j) \quad (3)$$

Equation (1) guarantees that each job is (fractionally) placed exactly once; Equation (2) ensures that each machine receives at most a load of T ; and due to Equation (3) jobs are only placed on eligible machines. We add additional constraints that have to be satisfied by any integral solution. In particular, we add the following constraints using parameters $UB(\ell, r)$ for each $\ell, r \in \mathcal{M}$ with $\ell \leq r$, which will be properly introduced shortly:

$$\sum_{j \in \mathcal{L} \cup \mathcal{H}} x_{ij} \leq 1 \quad \forall i \in \mathcal{M} \quad (4)$$

$$\sum_{i \in \mathcal{M}(\ell, r)} \sum_{j \in \mathcal{H}} x_{ij} \leq UB(\ell, r) \quad \forall \ell, r \in \mathcal{M}, \ell \leq r \quad (5)$$

Equation (4) captures the simple fact that no machine may receive more than one job of size larger than $0.5T$ and was used in [5] as well. The bound $UB(\ell, r)$, on the other hand, is defined in relation to the total load of small jobs that has to be scheduled in the respective interval $\mathcal{M}(\ell, r)$. In particular, we consider the overall load of small jobs that have to be placed in the interval together with the load due to huge jobs with their sizes rounded down to their minimum size. The respective load has to be bounded by T times the number of machines in the interval, i.e., $\sum_{i \in \mathcal{M}(\ell, r)} \sum_{j \in \mathcal{H}} (0.5 + \xi) T x_{ij} + p(\mathcal{S}(\ell, r)) \leq T |\mathcal{M}(\ell, r)|$. Since the number of huge jobs placed in an interval is integral for an integral solution, we can therefore set $UB(\ell, r) = \lfloor (T |\mathcal{M}(\ell, r)| - p(\mathcal{S}(\ell, r))) / ((0.5 + \xi) T) \rfloor$. We note that a constraint similar to Equation (5) is also used in [16, 17]. Summing up, we try to solve the linear program given by Equations (1)–(5) which is indeed a relaxation for RAI. If this is not successful, we reject T and otherwise round the solution x using the procedure described in the following and yielding a rounded solution \bar{x} .

Placement of Huge Jobs. Starting with the first machine in the ordering, we place the huge jobs as follows:

- Let i^* be the current machine and H the set of huge jobs that have not been placed yet and are eligible on i^* .
- If $\lfloor \sum_{i \in \mathcal{M}(0, i^*)} \sum_{j \in \mathcal{H}} x_{ij} \rfloor > \lfloor \sum_{i \in \mathcal{M}(0, i^* - 1)} \sum_{j \in \mathcal{H}} x_{ij} \rfloor$ and $H \neq \emptyset$, place a least flexible job $j \in H$ on i^* , i.e., we set $\bar{x}_{i^* j} = 1$.
- Consider the next machine in the ordering or stop if there is none.

This procedure indeed works and we preserve a connection to the original LP solution:

► **Lemma 2.** *All of the huge jobs are placed (on eligible machines) by the above procedure and, for each $\ell, r \in \mathcal{M}$ with $\ell \leq r$, we have $\sum_{i \in \mathcal{M}(\ell, r)} \sum_{j \in \mathcal{H}} \bar{x}_{ij} \leq \lceil \sum_{i \in \mathcal{M}(\ell, r)} \sum_{j \in \mathcal{H}} x_{ij} \rceil$.*

We note that this first rounding step is very similar to the first rounding step in [17].

Mapping out the Regions. In the next step, we divide the machines into regions, where each region did receive fractional large load of (roughly) one. To that end, we define a set of border machines \mathcal{B} as the machines considered from left to right where the sum of

fractionally placed large jobs hits a new integer, i.e., $\mathcal{B} = \{i' \in \mathcal{M} \mid [\sum_{i \in \mathcal{M}(0,i')} \sum_{j \in \mathcal{L}} x_{ij}] > [\sum_{i \in \mathcal{M}(0,i'-1)} \sum_{j \in \mathcal{L}} x_{ij}]\}$. Moreover, let $\mathcal{B} = \{i_1, \dots, i_q\}$ with $i_1 < \dots < i_q$ and i_0 the left-most machine with $\sum_{j \in \mathcal{L}} x_{i_0 j} > 0$. For each $s \in [q] = \{0, \dots, q-1\}$, we may initially define the s -th region as $R^s = \mathcal{M}(i_s, i_{s+1})$. At this point consecutive regions overlap by one machine. We want to change this, while guaranteeing that each region retains at least one *candidate machine* that may receive a large job in the following. In particular, a machine $i \in \mathcal{M}$ is a candidate if it did receive some fractional large or huge job in the LP solution, i.e., $\sum_{j \in \mathcal{H} \cup \mathcal{L}} x_{ij} > 0$, but no huge job afterwards, i.e., $\sum_{j \in \mathcal{H}} \bar{x}_{ij} = 0$. We denote the set of candidate machines as \mathcal{C} . For each $s \in [q-1]$, we apply the following procedure in incremental order:

- Check whether region R^s needs the last machine to have at least one candidate, i.e., $\mathcal{M}(i_s, i_{s+1} - 1) \cap \mathcal{C} = \emptyset$.
 - If this is the case, we set $R^{s+1} = \mathcal{M}(i_{s+1} + 1, i_{s+2})$ and otherwise set $R^s = \mathcal{M}(i_s, i_{s+1} - 1)$.
- After applying this procedure, we have:

► **Lemma 3.** *The regions are non-overlapping and each contain at least one candidate.*

Before proceeding with the placement of the large jobs, we note the following technical observation:

► **Lemma 4.** *Let $\ell, r \in \mathcal{M}$ with $\ell \leq r$, $\ell \in R^s$, $r \in R^t$, $k = |\{s, \dots, t\}|$, and $\text{fracLarge} = \sum_{i \in \mathcal{M}(\ell, r)} \sum_{j \in \mathcal{L}} x_{ij}$. Then we have $k - 2 < \text{fracLarge} < k + 2$. Furthermore, $\text{fracLarge} < k + 1$ if either $\ell > i_s$ or $r < i_{t+1}$ and $\text{fracLarge} < k$ if both of these conditions hold.*

Placement of Large Jobs. Using the regions, we place the large jobs via the following procedure starting with the first region:

- Let R^* be the current region and L the set of large jobs that have not been placed yet and are eligible on at least one candidate machine from R^* .
- Do the following *twice*: Pick a least flexible large job $j \in L$, place it on the leftmost eligible candidate machine $i \in R^*$, i.e. $\bar{x}_{ij} = 1$, and update L .
- Consider the next region in the ordering or stop if there is none.

Observe that the placement of both the large and huge jobs guarantees that only machines that did receive fractional large or huge load in the LP solution may receive any large or huge job and each such machine receives at most one such job. We argue that this procedure works and also retains some connection to the original LP solution x .

► **Lemma 5.** *All large jobs are placed (on eligible machines) by the described procedure and, for each $\ell, r \in \mathcal{M}$ with $\ell \leq r$, we have $\sum_{i \in \mathcal{M}(\ell, r)} \sum_{j \in \mathcal{L}} \bar{x}_{ij} < 2(\sum_{i \in \mathcal{M}(\ell, r)} \sum_{j \in \mathcal{L}} x_{ij} + 2)$.*

Proof. Regarding the second statement note that we place at most 2 jobs in each region and hence Lemma 4 directly yields the proof. As usual, we proof the first statement by contradiction. To that end, assume that there exists a large job j^* that is not placed by the procedure. First note, that there is at least one eligible candidate machine for j^* . To see this, consider the set M of eligible machines $i \in \mathcal{M}(j)$ that either received fractional load of j^* or some huge load, i.e., $x_{ij} > 0$ for $j \in \{j^*\} \cup \mathcal{H}$. Then Equation (4) implies $\sum_{i \in M} \sum_{j \in \mathcal{H}} x_{ij} \leq |M| - 1$. Hence, at most $|M| - 1$ many huge jobs are placed on machines from M due to Lemma 2 and therefore at least one of these machines is a candidate. There are two possibilities why j^* was not placed on such a machine: either a less flexible job got placed on the machine, or two other less flexible jobs were already placed in the same region. Let $r^* = r(j^*)$ and $\ell^* \leq \ell$ be minimal with the property that each large job that was placed

in $\mathcal{M}(\ell^*, r^*)$ is less flexible than j^* and each free candidate machine in the interval is free because two other machines in the same respective region already received a large job less flexible than j^* . Furthermore, let J^* be the set of large jobs placed in $\mathcal{M}(\ell^*, r^*)$ together with j^* . We argue that $\ell(j) \geq \ell^*$ for each $j \in J^*$. Otherwise, there exists a job $j \in J^*$ eligible on machine $\ell^* - 1$. Then there are three possibilities regarding this machine. It was not a candidate before the procedure; it was a candidate and received a job less flexible than j (and therefore also less flexible than j^*); or it was a candidate and did not receive a large job because two other machines in the same region received a job less flexible than j . Each yields a contradiction to the definition of ℓ^* . Let $\text{fracLarge} = \sum_{i \in \mathcal{M}(\ell^*, r^*)} \sum_{j \in \mathcal{L}} x_{ij}$ be the sum of fractional large jobs in $\mathcal{M}(\ell^*, r^*)$ according to x . Note that we did show $J^* \subseteq \mathcal{J}(\ell^*, r^*)$ and hence $\text{fracLarge} \geq |J^*|$.

Let $M^* \subseteq \mathcal{M}(\ell^*, r^*)$ be the set of machines that did receive a fraction of a job from $J^* \cup \mathcal{H}$. Then Equation (4) implies $\sum_{i \in M^*} \sum_{j \in \mathcal{H}} x_{ij} \leq |M^*| - |J^*|$, and furthermore Lemma 2 yields that at most $|M^*| - |J^*|$ huge jobs are placed on machines from $|M^*|$. Hence, there are at least $|J^*|$ candidate machines in $\mathcal{M}(\ell^*, r^*)$. Since not all of the jobs from J^* have been placed by the procedure, there is therefore at least one free machine in $i^* \in \mathcal{M}(\ell^*, r^*)$. The definition of ℓ^* yields, that two jobs less flexible than j^* have been placed in the same region as i^* and these jobs have to be included in J^* (and the machines they are placed on in $\mathcal{M}(\ell^*, r^*)$).

We now take a closer look at the regions (partially) included in $\mathcal{M}(\ell^*, r^*)$. Let $\ell^* \in R^s$, $r^* \in R^t$, and $k = |\{s, \dots, t\}|$. We consider three cases: If we have $\ell^* = i_s$ and $r^* = i_{t+1}$, i.e., the borders of the interval correspond to the (original) outer borders of their regions, then each of the regions R^s, \dots, R^t did receive at least one job from J^* and one received at least two yielding $k \leq |J^*| - 2 \leq \text{fracLarge} - 2$. Moreover, if $\ell^* > i_s$ or $r^* < i_{t+1}$, then one of the regions R^s, \dots, R^t may not have received a job from J^* changing the inequality to $k \leq \text{fracLarge} - 1$. Lastly, if both $\ell^* > i_s$ and $r^* < i_{t+1}$, then the two outer regions may have received no job from J^* yielding $k \leq \text{fracLarge}$. However, Lemma 4 considers the same three cases, yielding $\text{fracLarge} < k + 2$, $\text{fracLarge} < k + 1$, and $\text{fracLarge} < k$, respectively. ζ

Placement of Small Jobs. Lastly we place the small jobs. Starting with the first machine, we do the following:

- Let i^* be the current machine and J the set of jobs that have not been placed yet and are eligible on i^* .
- Successively place least flexible jobs j on i^* , i.e., set $\bar{x}_{i^*j} = 1$, until either $J = \emptyset$ or placing the next job would raise the load of i^* above $(2 - \gamma)T$.
- Consider the next machine in the ordering or stop if there is none.

We argue that this procedure works under certain conditions:

► **Lemma 6.** *All small jobs are placed (on eligible machines) by the described procedure if $\gamma \leq \xi$, $\gamma + \xi \leq \frac{1}{12}$, and $8\xi + 7\gamma \leq 0.75$ hold. In the resulting schedule, each machine has a load of at most $(2 - \gamma)T$.*

Proof (Idea). The main idea of the proof is to assume that some job cannot be placed and to use this to construct some interval in which each machine received some minimum amount of load and in which each placed job had to be placed in the respective interval. Then Lemma 2, Lemma 5, and the constraints of the LP are used to show a contradiction. Depending on the number of large jobs in the interval, either Equation (2) or Equation (5) are critical. \blacktriangleleft

Lastly, we choose values for ξ and γ which satisfy all the requirements of the above lemma and maximize γ . The biggest γ is achieved by setting $\gamma = \xi = \frac{1}{24}$. This concludes the proof of Theorem 1.

3 Complexity Results

Remember that we use the notation $[n] = \{0, \dots, n-1\}$ for each integer n . The complexity result in this work directly build upon the ones in [12]. In that work, a satisfiability problem denoted as 3-SAT* was introduced and shown to be NP-hard, and all reductions in the present work start from this problem. An instance of the problem 3-SAT* is a conjunction of clauses with exactly 3 literals each. Each of the clauses is either a 1-in-3-clause or a 2-in-3-clause, that is, they are satisfied if exactly one or two of their literals, respectively, evaluate to *true* in a given truth assignment. We denote a k -in-3-clause with literals x , y , and z as $(x, y, z)_k$ and the truth values true and false are denoted as \top and \perp in the following. There are as many 1-in-3-clauses in a 3-SAT* instance as there are 2-in-3-clauses, and, furthermore, each literal occurs exactly twice. Hence, a minimal example for a 3-SAT* instance is given by $(x_0, x_1, \neg x_2)_1 \wedge (\neg x_0, x_1, x_2)_1 \wedge (x_0, \neg x_1, \neg x_2)_2 \wedge (\neg x_0, \neg x_1, x_2)_2$: We have two 1-in-3-clauses and two 2-in-3-clauses, and two occurrences of x_i and $\neg x_i$ for each $i \in [3]$. The formula is satisfied if we map every variable to \perp .

In each reduction, we start with an instance I of 3-SAT* with m many 1-in-3-clauses C_0, \dots, C_{m-1} , m many 2-in-3-clauses C_m, \dots, C_{2m-1} and n variables x_0, \dots, x_{n-1} . Since there are $2m$ clauses with 3 literals each and 4 occurrences for each variable, we have $6m = 4n$. In the following, the precise positions of the occurrences of the variables are important and we have to make them explicit. To this end, let for each $j \in [n]$ and $t \in [4]$ the pair (j, t) correspond to the first or second positive occurrence of variable x_j if $t = 0$ or $t = 1$, respectively, and to the first or second negative occurrence of variable x_j if $t = 2$ or $t = 3$. Furthermore, let $\kappa : [n] \times [4] \rightarrow [2m] \times [3]$ be the bijection that maps (j, t) to the corresponding clause index and position in that clause. For instance, in the above example we have $\kappa(0, 2) = (1, 0)$ and $\kappa(2, 1) = (3, 2)$.

Next, we construct an instance I' of the problem considered in the respective case. For the restricted assignment type problems, all job sizes are integral and upper bounded by some constant T such that the overall size of the jobs equals $|\mathcal{M}|T$. Hence, if a machine receives jobs with overall size more or less than T , the objective function value is worse than T for both the makespan and fair allocation case. The goal is to show, that there is a schedule with makespan T for I' , if and only if I is a yes-instance. This rules out approximation algorithms with rate smaller than $(T+1)/T$ for the makespan problem, and with rate smaller than $T/(T-1)$ for the fair allocation variant since the overall load is $|\mathcal{M}|T$. For the low rank problem, we first design a restricted assignment reduction using the above approach and then show that there exist low rank scheduling instances that approximate the restricted assignment instance with arbitrary precision.

Simple Reduction. We start with a simple reduction for the general restricted assignment problem (with arbitrary restrictions) introducing several ideas and gadgets relevant for all of the following reductions. Note that the reduction is very similar to the one by Ebenlendr et al. [5] and to a reduction in [12].

We have three types of basic jobs and machines, namely, truth assignment machines and jobs that are used to assign truth values to variables, clause machines and jobs that model clauses being satisfied, and variable jobs that connect the first two types:

■ **Table 1** Resource demands and capacities of the jobs and machines, respectively, for the case with 3 resources.

Job/Mach.	Res. 1	Res. 2	Res. 3
$\text{TMach}(j, 0)$	$4j + 1$	$4n - 4j$	1
$\text{TMach}(j, 1)$	$4j + 3$	$4n - 4j$	0
$\text{TJob}(j)$	$4j$	$4n - 4j$	0
$\text{CMach}(i, s), \kappa^{-1}(i, s) = (j, t)$	$4j + t$	$4n - (4j + t)$	$2 + i$
$\text{CJob}(i, s)$	0	0	$2 + i$
$\text{VJob}(j, t)$	$4j + t$	$4n - (4j + t)$	$1 - \lfloor \frac{t}{2} \rfloor$

- There are truth assignment machines $\text{TMach}(j, q)$ with $j \in [n]$ and $q \in [2]$ and one truth assignment job $\text{TJob}(j)$ with size 2 and eligible on $\{\text{TMach}(j, 0), \text{TMach}(j, 1)\}$.
- There are clause machines $\text{CMach}(i, s)$ for each $i \in [2m]$ and $s \in [3]$ and three clause jobs $\text{CJob}(i, s)$ each eligible on $\{\text{CMach}(i, s') \mid s' \in [3]\}$. The job $\text{CJob}(i, 0)$ has size 1, $\text{CJob}(i, 2)$ has size 2, and $\text{CJob}(i, 1)$ has size 2 if clause C_i is a 1-in-3-clause and size 1 otherwise.
- Lastly, there are variable jobs $\text{VJob}(j, t)$ for each $j \in [n]$ and $t \in [4]$ each of size 1 and eligible on $\{\text{TMach}(j, \lfloor \frac{t}{2} \rfloor), \text{CMach}(\kappa(j, t))\}$.

First note:

▷ **Claim 7.** The overall job size $\sum_{j \in \mathcal{J}} p(j)$ is equal to $2|\mathcal{M}|$.

Consider the case that we have a satisfying truth assignment for instance I . If variable x_j is assigned to \top , we place $\text{TJob}(j)$ on $\text{TMach}(j, 0)$, $\text{VJob}(j, 0)$ and $\text{VJob}(j, 1)$ on $\text{CMach}(\kappa(j, 0))$ and $\text{CMach}(\kappa(j, 1))$, respectively, together with local size 1 clause jobs. Furthermore, $\text{VJob}(j, 2)$ and $\text{VJob}(j, 3)$ are placed on $\text{TMach}(j, 1)$ and $\text{CMach}(\kappa(j, 2))$ and $\text{CMach}(\kappa(j, 3))$ each receive a local size 2 clause job. If variable x_j is assigned to \perp , we place $\text{TJob}(j)$ on $\text{TMach}(j, 1)$, and the placement strategy of the positive and negative variable jobs is reversed. Note that the placement of the clause jobs has to work out since the truth assignment is satisfying. This approach yields a schedule with makespan 2. However, it is also easy to see that a schedule with makespan 2 yields a satisfying truth assignment by basing the assignment of x_j on the placement of $\text{TJob}(j)$, and hence we have:

► **Lemma 8.** *There is a satisfying truth assignment for I , if and only if there is a schedule with makespan 2 for I' .*

This reduction can be considered the basis of all the other ones considered in this work.

3.1 Three Resources

In the RAR(3) case, we can use essentially the same construction as above. However, the sets of eligible machines are defined using the resources and are slightly different. The resource demands and capacities are specified in Table 1. The choice of resources implies:

▷ **Claim 9.** We have $\mathcal{M}(\text{TJob}(j)) = \{\text{TMach}(j, 0), \text{TMach}(j, 1)\}$ for each $j \in [n]$ and $\mathcal{M}(\text{VJob}(j, t)) = \{\text{TMach}(j, \lfloor \frac{t}{2} \rfloor), \text{CMach}(\kappa(j, t))\}$ for each $j \in [n]$ and $t \in [4]$.

Hence, the truth assignment and variable jobs have the same sets of eligible machines as before. For the clause jobs this is not true, however, a similar claim holds. We call a schedule that assigns a load of exactly T to each machine a T -schedule. Using the fact, that in a 2-schedule each clause machine has to receive at least one clause job, it is easy to show the following:

77:10 Interval, Resource Restricted, and Low Rank Scheduling

▷ **Claim 10.** In any 2-schedule each machine from $\{\text{CMach}(i, s) \mid s \in [3]\}$ receives exactly one job from $\{\text{CJob}(i, s) \mid s \in [3]\}$.

Hence, Lemma 8 works the same as before and we have:

► **Theorem 11.** *There is no better than 1.5-approximation for RAR(3) and no better than 2-approximation for the fair allocation version of this problem, unless $P=NP$.*

3.2 Interval Restrictions

In order to motivate the new ideas for the RAI reduction and to make them easier to understand, it is helpful to revisit the reduction from [12] first. One of the main ingredients in that result is a simple trick that we will also use extensively.

Pyramid Trick. Consider the following setting: We have 2ℓ consecutive machines and ℓ pairs of jobs. The i -th pair of jobs is eligible on the i -th machine and up to and including the $(2\ell + 1 - i)$ -th machine. Furthermore, we assume that each machine has to receive at least one of the jobs. Then the first and last machine each have to receive one job from the first pair because there are no other eligible jobs that can be processed on these machines. Now, the same argument can be repeated for the second and second to last machine and so on. Hence, machine i and $(2\ell + 1 - i)$ each have to receive exactly one job from pair i .

Sorting. Next, consider that in the ordering of the machines the truth assignment machines are placed on the left and the clause machines on the right. We could use similar truth assignment and clause jobs as in the reduction in the beginning of this chapter. However, variable jobs each are eligible on one truth assignment and one clause machine. Hence they have to be eligible on all machines in between in a naive adaptation of the reduction to the interval case. If we want to use the pyramid trick to deal with this problem, then intuitively decisions regarding variables in later clauses have to be made before the decisions for variables in earlier clauses and this, of course, cannot be guaranteed regardless of the fixed order of the clauses or variables. The main work in [12] was to remedy this situation by – roughly speaking – sorting the information regarding the variables made in the truth assignment gadget to enable the use of the pyramid trick. To do so several gadgets have been introduced that were intertwined with the truth assignment gadget and carefully build up the ordered information using the pyramid trick and interlocking job sizes. The problem with this approach is that the job sizes can get big rather fast if too many different job types are eligible on the same machines resulting in a high value for T . Now, the main idea in the present work is to decouple the decision and the sorting process and make the sorting process as simple as possible to enable smaller job sizes and therefore a stronger result. Curiously, the sorting process in [12] could be interpreted as some variant of insertion sort, while the one used in the present reduction resembles bubble sort.

Machines and Order. Let $k \in \mathcal{O}(n^2)$ be a parameter to be specified later in this paragraph. In addition to the truth assignment and clause machines, we introduce the following ones:

- For each $j \in [n]$ and $t \in [4]$ there are two gateway machines: one forward $\text{FGMach}(j, t)$ and one backward gateway machine $\text{BGMach}(j, t)$.
- For each $\ell \in [k]$, $j \in [n]$, and $t \in [4]$ there are two sorting machines: one forward $\text{FSMach}(\ell, j, t)$ and one backward sorting machine $\text{BSMach}(\ell, j, t)$.

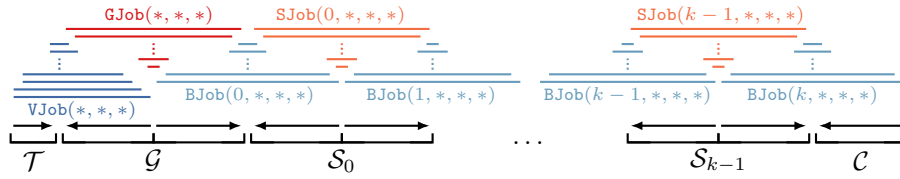
Let \mathcal{T} , \mathcal{G} , and \mathcal{C} be the sets of truth assignment, gateway, and clause machines, respectively. Moreover, for each $\ell \in [k]$ let $\mathcal{S}_\ell = \{\text{BSMach}(\ell, j, t), \text{FSMach}(\ell, j, t) \mid j \in [n], t \in [4]\}$ be the ℓ -th set of sorting machines. We define the overall order of the machines by setting an internal order for each set of machines as well as an order of the machine sets. However, before we can do so, we need some additional concepts and notation. In particular, let φ_0 be the sequence of (j, t) -pairs with $j \in [n]$, $t \in [4]$ with increasing lexicographical order and $\psi_0 = \kappa(\varphi_0)$, i.e., $\varphi_0 = ((0, 0), \dots, (0, 3), \dots, (n-1, 0), \dots, (n-1, 3))$ and $\psi_0 = (\kappa(0, 0), \dots, \kappa(0, 3), \dots, \kappa(n-1, 0), \dots, \kappa(n-1, 3))$. Hence, ψ_0 is a permutation of the pairs (i, s) with $i \in [2m]$ and $s \in [3]$. We consider sorting ψ_0 with the goal of reaching the increasing lexicographical order. Let k be the number of transpositions performed by bubble sort if we do this. Furthermore, let $\psi_{\ell+1}$ for $\ell \in [k]$ be the sequence we get after the first $(\ell+1)$ -transpositions and $\varphi_{\ell+1} = \kappa^{-1}(\psi_{\ell+1})$. The use of bubble sort guarantees that $k \in \mathcal{O}(n^2)$ and that two consecutive sequences $\varphi_\ell, \varphi_{\ell+1}$ differ only by two consecutive entries that are transposed. For any finite sequence χ , we denote the reversed sequence as $\bar{\chi}$. Now, the ordering is specified as follows:

- The sets are ordered as follows: $\mathcal{T}, \mathcal{G}, \mathcal{S}_0, \dots, \mathcal{S}_{k-1}, \mathcal{C}$.
- The truth assignment machines are ordered in *increasing* lexicographical order of the indices (j, q) .
- The clause machines are ordered in *decreasing* lexicographical order of the indices (i, s) .
- The backward gateway machines are placed before the forward gateway machines and for each $\ell \in [k]$ the backward sorting machines are placed before the forward sorting machines from \mathcal{S}_ℓ as well.
- The forward and backward gateway machines are ordered in increasing and decreasing lexicographical order of the indices (j, t) , i.e., φ_0 and $\bar{\varphi}_0$, respectively.
- For each $\ell \in [k]$, the backward sorting machines are sorted according to the placement of the (j, t) indices in $\bar{\varphi}_\ell$.
- For each $\ell \in [k]$, the forward sorting machines are sorted according to the placement of the (j, t) indices in $\varphi_{\ell+1}$.

The peculiar ordering of the machines is designed to enable the use of the pyramid trick.

Jobs, Sizes, and Eligibilities. We give a full list of all jobs together with their sizes and define the sets of eligible machines by stating the respective first and last eligible machine for each job. We will need one more definition: Let $\xi : [k] \rightarrow [n] \times [4]$ be the function that maps ℓ to the distinct pair (j, t) that has a higher index in $\varphi_{\ell+1}$ than in φ_ℓ .

- Truth assignment jobs: For each $j \in [n]$ there is a job $\text{TJob}(j)$ with size 2, first machine $\text{TMach}(j, 0)$ and last machine $\text{TMach}(j, 1)$.
- Variable jobs: For each $j \in [n]$, $t \in [4]$, and $\circ \in \{\top, \perp\}$ there is a job $\text{VJob}(j, t, \circ)$ with size 2 if $\circ = \perp$ and 3 otherwise, first machine $\text{TMach}(j, \lfloor t/2 \rfloor)$ and last machine $\text{BGMach}(j, t)$.
- Gateway jobs: For each $j \in [n]$, $t \in [4]$, and $\circ \in \{\top, \perp\}$ there is a job $\text{GJob}(j, t, \circ)$ with size 5 if $\circ = \perp$ and 4 otherwise, first machine $\text{BGMach}(j, t)$ and last machine $\text{FGMach}(j, t)$.
- Bridge jobs: For each $\ell \in [k+1]$, $j \in [n]$, $t \in [4]$, and $\circ \in \{\top, \perp\}$ there is a bridge job $\text{BJob}(\ell, j, t, \circ)$ with size 1 if $\circ = \perp$ and 2 otherwise, first machine either $\text{FGMach}(j, t)$ if $\ell = 0$ or $\text{FSMach}(\ell-1, j, t)$ otherwise, and last machine either $\text{BSMach}(\ell, j, t)$ if $\ell < k$ or $\text{CMach}(\kappa(j, t))$ otherwise.
- Sorting jobs: For each $\ell \in [k]$, $j \in [n]$, $t \in [4]$, and $\circ \in \{\top, \perp\}$ there is a job $\text{SJob}(\ell, j, t, \circ)$. If $\xi(\ell) = (j, t)$, it has size 4 if $\circ = \perp$ and 3 otherwise, and, if $\xi(\ell) \neq (j, t)$, it has size 7 if $\circ = \perp$ and 6 otherwise. The first machine of $\text{SJob}(\ell, j, t, \circ)$ is $\text{BSMach}(\ell, j, t)$ and the last machine is $\text{FSMach}(\ell, j, t)$.



■ **Figure 1** A visualization of the job and machine structure for the reduction regarding RAI. The lower part corresponds to sets of machines with arrows corresponding to the direction of their ordering; and the upper part to different job sets with lines corresponding to intervals of eligible machines for pairs of jobs. Truth assignment and clause jobs as well as private loads are not depicted.

- Clause jobs: For each $i \in [2m]$ and $s \in [3]$ there is a job $\text{CJob}(i, s)$ with size 7 if $s = 1$, $8 - k$ if $s = 2$ and C_i is a k -in-3-clause, and 6 if $s = 3$, first machine $\text{CMach}(i, 2)$ and last machine $\text{CMach}(i, 0)$.
- Private loads: Each truth assignment machine has a private load (a job eligible only one one machine) of 2, each backward or forward gateway machine a load of 1 or 2, respectively, and for each $\ell \in [k]$ the sorting machines $\text{BSMach}(\ell, \xi(\ell))$ and $\text{FSMach}(\ell, \xi(\ell))$ have a private load of 3.

Using this reduction, we can show:

- **Theorem 12.** *There is no better than $\frac{9}{8}$ -approximation for RAI and no better than $\frac{8}{7}$ -approximation for the fair allocation version of this problem, unless $P=NP$.*

While the formal proof of this result exceeds the scope of this short version of the paper, we briefly discuss the main idea: We have a truth assignment gadget that determines the truth values of the variables and is followed by the gateway gadget, whose sole purpose is to decouple the used job sizes in the truth assignment gadget and the sorting gadget. Next there is the sorting gadget that slowly reorders the information about the decisions in the truth assignment gadget, and lastly there is the clause gadget in which the truth assignment is evaluated. The connection between the truth assignment and gateway gadget is provided by the variable jobs and all other connections are realized via bridge jobs. A sketch of the overall structure of the reduction is provided in Figure 1.

4 Conclusion

We conclude this work with a brief discussion of possible future research directions. There are some obvious questions that can be pursued directly building upon the presented results, i.e., a better approximation ration for RAI or even stronger inapproximability results for RAI or RAR(2). Of course, an improved approximation ratio for any problem of the family would be interesting to develop. We would like to highlight LRS(2), in particular, as the in some sense easiest problem in the family without a known polynomial time approximation with ratio better than 2. Lastly, only very little is known regarding fixed-parameter tractable algorithms for this family of problems. For instance, it is open whether RAR(1) is fixed-parameter tractable with respect to the objective value.

References

- 1 Aditya Bhaskara, Ravishankar Krishnaswamy, Kunal Talwar, and Udi Wieder. Minimum makespan scheduling with low rank processing times. 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 937–947. SIAM, 2013. doi:10.1137/1.9781611973105.67.

- 2 Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. On $(1, \varepsilon)$ -restricted assignment makespan minimization. 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 1087–1101. SIAM, 2015. doi:10.1137/1.9781611973730.73.
- 3 Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPICs*, pages 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.STACS.2017.22.
- 4 Lin Chen, Deshi Ye, and Guochuan Zhang. An improved lower bound for rank four scheduling. *Oper. Res. Lett.*, 42(5):348–350, 2014. doi:10.1016/j.orl.2014.06.003.
- 5 Tomáš Ebenlendr, Marek Krcál, and Jirí Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68(1):62–80, 2014. doi:10.1007/s00453-012-9668-9.
- 6 Leah Epstein and Asaf Levin. Scheduling with processing set restrictions: Ptas results for several variants. *International Journal of Production Economics*, 133(2):586–595, 2011. doi:10.1016/j.ijpe.2011.04.024.
- 7 Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.
- 8 Klaus Jansen, Marten Maack, and Roberto Solis-Oba. Structural parameters for scheduling with assignment restrictions. *Theor. Comput. Sci.*, 844:154–170, 2020. doi:10.1016/j.tcs.2020.08.015.
- 9 Kamyar Khodamoradi, Ramesh Krishnamurti, Arash Rafiey, and Georgios Stamoulis. PTAS for ordered instances of resource allocation problems with restrictions on inclusions. *CoRR*, abs/1610.00082, 2016. arXiv:1610.00082.
- 10 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990. doi:10.1007/BF01585745.
- 11 Chung-Lun Li and Xiuli Wang. Scheduling parallel machines with inclusive processing set restrictions and job release times. *Eur. J. Oper. Res.*, 200(3):702–710, 2010. doi:10.1016/j.ejor.2009.02.011.
- 12 Marten Maack and Klaus Jansen. Inapproximability results for scheduling with interval and resource restrictions. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPICs*, pages 5:1–5:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.STACS.2020.5.
- 13 Marten Maack, Simon Pukrop, and Anna Rodriguez Rasmussen. (in-)approximability results for interval, resource restricted, and low rank scheduling. *CoRR*, abs/2203.06171, 2022. doi:10.48550/arXiv.2203.06171.
- 14 Gabriella Muratore, Ulrich M. Schwarz, and Gerhard J. Woeginger. Parallel machine scheduling with nested job assignment restrictions. *Oper. Res. Lett.*, 38(1):47–50, 2010. doi:10.1016/j.orl.2009.09.010.
- 15 Petra Schuurman and Gerhard J Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999. doi:10.1002/(SICI)1099-1425(199909/10)2:5<203::AID-JOS26>3.0.CO;2-5.
- 16 Ulrich M. Schwarz. *Approximation algorithms for scheduling and two-dimensional packing problems*. PhD thesis, University of Kiel, 2010. URL: http://eldiss.uni-kiel.de/macau/receive/dissertation_diss_00005147.
- 17 Chao Wang and René Sitters. On some special cases of the restricted assignment problem. *Inf. Process. Lett.*, 116(11):723–728, 2016. doi:10.1016/j.ipl.2016.06.007.
- 18 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

Localized Geometric Moves to Compute Hyperbolic Structures on Triangulated 3-Manifolds

Clément Maria   

Inria Sophia Antipolis-Méditerranée, France

Owen Rouillé  

Inria Sophia Antipolis-Méditerranée, France

Abstract

A fundamental way to study 3-manifolds is through the geometric lens, one of the most prominent geometries being the hyperbolic one. We focus on the computation of a complete hyperbolic structure on a connected orientable hyperbolic 3-manifold with torus boundaries. This family of 3-manifolds includes the knot complements.

This computation of a hyperbolic structure requires the resolution of gluing equations on a triangulation of the space, but not all triangulations admit a solution to the equations.

In this paper, we propose a new method to find a triangulation that admits a solution to the gluing equations, using convex optimization and localized combinatorial modifications. It is based on Casson and Rivin’s reformulation of the equations. We provide a novel approach to modify a triangulation and update its geometry, along with experimental results to support the new method.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases knots and 3-manifolds, triangulation, hyperbolic structure, Thurston equations

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.78

Related Version *Full Version:* <https://arxiv.org/abs/2112.06360>

Funding *Clément Maria:* Partially funded by ANR project ANR-20-CE48-0007 (AlgoKnot).

Owen Rouillé: Fully funded by ANR project ANR-20-CE48-0007 (AlgoKnot).

1 Introduction

A main problem of knot theory is to tell whether two knots are equivalent or distinct. Equivalence between knots is defined by the existence of an isotopy of the ambient space that would turn one knot into the other, i.e., a continuous deformation of the space that preserves the entanglement.

Isotopies are too difficult to compute in practice, and practitioners use *invariants* to tackle the knot equivalence problem. A *topological invariant* is a quantity assigned to a presentation of a knot, that is invariant by isotopy.

An important family of knots are the *hyperbolic knots*, which are the knots whose complements admit a *complete hyperbolic metric*. They are the subjects of active mathematical research, which motivates the introduction of efficient algorithmic tools to study their geometric properties, and most notably their *hyperbolic volume*. The hyperbolic volume of a hyperbolic knot is a topological invariant which is powerful at distinguishing between non-equivalent knots, is non-trivial to compute, and is at the heart of several deep conjectures in topology [11].

If it exists, the complete hyperbolic metric on a 3-manifold is unique[10], and may be combinatorially represented by a *complete hyperbolic structure* (CHS). In order to compute geometric properties (such as volume) of a hyperbolic knot, one triangulates the knot complement, and try to assign hyperbolic shapes to its tetrahedra. If these *shapes* verify a set of non-linear constraints called the *gluing equations*, they form a CHS and encode the



© Clément Maria and Owen Rouillé;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 78; pp. 78:1–78:13

Leibniz International Proceedings in Informatics

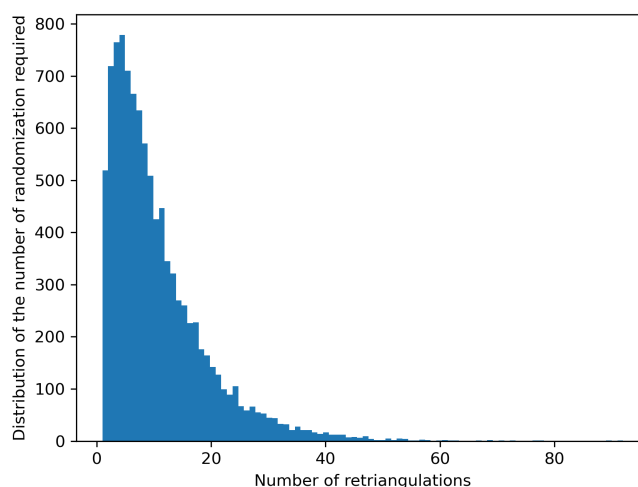


LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

78:2 Localized Geometric Moves to Compute Hyperbolic Structures

■ **Table 1** On all ~ 9.7 millions prime knots with crossing numbers ranging from 12 to 17, alternating and non-alternating, we indicate (*% Failure on first try*) the percentage of knot complements (after triangulation and simplification) on which **SnapPy** fails to compute a CHS on first try. We also indicate (*Highest expected nb of retriang.*) the highest, over all knots, expected number of random re-triangulations necessary for **SnapPy** to succeed finding a CHS.

	Alternating						Non-Alternating					
#crossings	12	13	14	15	16	17	12	13	14	15	16	17
% Failure on first try	0.9	1.6	2.4	3.3	4.4	5.7	0.8	0.7	1.2	1.7	2.4	3.3
Expected nb of retriang.	2.9	3.9	6.2	7.1	9.5	15.9	2.0	3.1	6.1	11.8	9.7	13.5

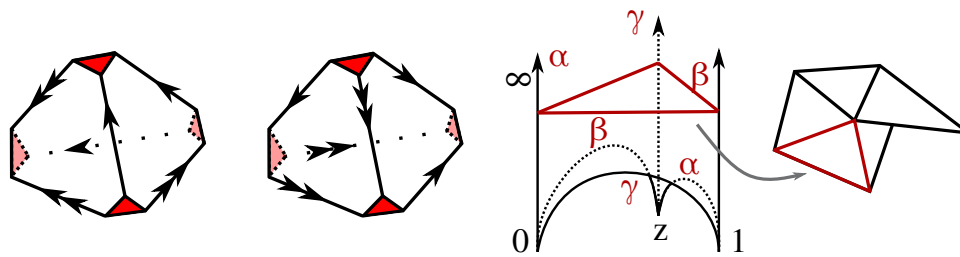


■ **Figure 1** Distribution of the number of randomizations required to find a CHS for the complement of the knot “17nh_2654001” of the census, for 10000 tries. The mean is 10.3 randomizations, and the standard deviation is 8.7. The minimum number of re-triangulations is 1, and the maximum 92.

complete hyperbolic metric of the space. A major issue with this approach is that a solution to the constraints may not exist on all triangulations of a manifold, even if, as topological object, the manifold can carry a complete hyperbolic metric. Worst, it is not known whether every hyperbolic 3-manifold admits a triangulation on which a solution as CHS exists.

In practice, given an input knot, software can construct a triangulation of the knot complement, and simplify it to have a CHS. But if this fails, the only implemented practical solution, proposed by **SnapPy**[3], is to randomly modify and simplify the triangulation before trying again until a CHS is found.

Table 1 provides data on the search for a CHS with **SnapPy**, on the ~ 9.7 millions prime knots with crossing numbers up to 17. As observed, **SnapPy** has a high rate of success in finding a CHS after a standard triangulation and simplification of the knot complement. However, this standard construction of a triangulation fails to admit a CHS on more than 350 thousand knots in the census, and the percentage of problematic triangulations needing re-triangulations tends to increase with the number of crossings. Additionally, we observe that some knot complements may require in expectation a high number of random re-triangulations (up to 15.9), and the number of re-triangulations may itself suffer a high variance, as illustrated in Figure 1.



■ **Figure 2** Left: ideal triangulation of the complement of the 8-knot with two tetrahedra. The ideal vertex is truncated, and the red surface gives the torus link of the ideal vertex after gluing of the tetrahedra following the edge identifications. Middle: ideal tetrahedron in the upper half-space model, the dihedral angles are denoted α , β and γ , the complex shape parameter z is associated to the edge between 0 and ∞ . Right: example of shearing singularity, each triangle corresponds to a ideal tetrahedron seen from above (gray arrow).

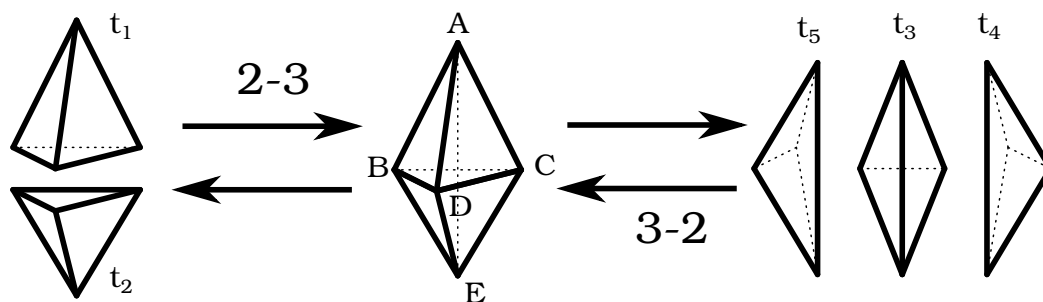
Checking for the existence of a CHS requires the resolution of the non-linear gluing equations (with, e.g., Newton optimization method). Reducing the number of re-triangulations is consequently critical for performance of computation in knot theory, most notably for knots on which the state-of-the-art methods implemented in `SnapPy` require large numbers of re-triangulations, and even more so when proceeding to very large scale experiments such as the computation of the knot censuses [1] that are of great use to practitioners, where “difficult” knots are many.

Contribution. This paper introduces a new heuristic based algorithm to improve on the random approach for re-triangulating. The method is inspired by Casson and Rivin’s reformulation of the gluing equations [15, 4]: the gluing equations are split into a linear part and a non-linear part, and the resolution reduces to a convex optimization problem on a polytope domain. If the triangulation does not admit a complete structure, the optimization problem will converge on the boundary of the polytope and we exploit this information to introduce new *localized moves* to modify combinatorially the triangulation while reusing the partially computed geometry. We introduce necessary background on triangulations and geometry in Section 2, and the computation of hyperbolic structures with optimization in Section 3. We analyze precisely the behavior of the optimization phase on triangulations not admitting a CHS in Section 4, and introduce a re-triangulation algorithm in Section 5 guided by the partially computed geometry of the optimization phase. We illustrate experimentally the interest of the approach in Section 6 and propose a hybrid method with `SnapPy` in Section 6.2, that outperforms the state-of-the-art.

Note that, in this article we focus on complements of hyperbolic knots. However, the techniques introduced extend to more general hyperbolic 3-manifolds with torus boundaries [4].

2 Background

In this article, we focus on knot complements, i.e., non-compact 3-manifolds obtained by removing a closed regular neighborhood of a knot K in the sphere S^3 .



■ **Figure 3** Illustration of the Pachner moves 2-3 and 3-2.

2.1 Generalized and ideal triangulations

A *generalized triangulation* T is a collection of n abstract tetrahedra whose triangular facets are identified (e.g. Figure 2 (left)), or *glued*, in pairs. Note that the facets of the same tetrahedron may be glued together, and generalized triangulations are more general than simplicial complexes. The link of a vertex in T is the frontier of a closed regular neighborhood, and is itself a closed triangulated surface embedded in the triangulation. If the link of a vertex v is a 2-sphere, we call v an *internal* vertex, otherwise (e.g., if the link is a torus), we call v an *ideal* vertex.

A 1-vertex ideal triangulation is a triangulation with exactly 1 ideal vertex, and no internal vertices. They represent non-compact 3-manifolds, that can be recovered from the 1-vertex ideal triangulation by considering their realization where the vertex has been removed. Every knot complement can be represented by a 1-vertex ideal triangulation, where the link of the ideal vertex gives the frontier of a closed regular neighborhood of the knot. Intuitively, this is a triangulation of the sphere S^3 where the knot has been shrunk into a single point, distorting its neighborhood. Such 1-vertex ideal triangulations of a knot complement can be computed in polynomial time [6, 5] from a planar drawing of a knot.

Any two ideal triangulations of the same 3-manifold can be connected by a sequence of *Pachner moves* [13]. For 1-vertex ideal triangulations, they consist of the moves 2-3 and 3-2, inverse of each other, pictured in Figure 3.

2.2 Combinatorial description of hyperbolic geometry

Certain topological 3-manifolds can be equipped uniformly with a complete hyperbolic metric, which is unique up to isometry. They are called *hyperbolic manifolds*. They include the vast and important family of complements of *hyperbolic knots*.

We use the upper half-space model to represent the hyperbolic space \mathbb{H}^3 . This representation corresponds to $\{(z, r) | z \in \mathbb{C}, r \in \mathbb{R}_+^*\}$ with $\partial\mathbb{H}^3 = \mathbb{C} \cup \{\infty\}$ (from now on denoted $\partial\mathbb{H}^3$) consisting of the bottom plane $\mathbb{C} \times \{0\}$, together with the point at infinity, where geodesics are arcs of circles orthogonal to $\partial\mathbb{H}^3$. This model is *conformal*, i.e., Euclidean angles in the upper half space give the values of the angles in the hyperbolic space.

An *ideal hyperbolic tetrahedron* is the hyperbolic convex hull of four distinct points of $\partial\mathbb{H}^3$. These points are the (ideal) vertices of the tetrahedron.

A vertex on $\partial\mathbb{H}^3$ is a vertex at infinity, thus it is not part of the tetrahedron. An example of ideal tetrahedron is shown in Figure 2 (middle). Up to isometry, the geometric shape of an ideal tetrahedron can be represented by a single complex number:

► **Definition 1** (Shape parameter). *Given an ideal hyperbolic tetrahedron, there exists an isometry sending three of its vertices to 0, 1, and ∞ , and ensuring that the fourth vertex has positive imaginary part in the complex plane in $\partial\mathbb{H}^3 = \mathbb{C} \cup \{\infty\}$. The coordinate z of this fourth vertex is the shape parameter of the tetrahedron.*

The shape parameter defines and illustrates the shape of an ideal tetrahedron. It depends on which vertices are sent to 0, 1, and ∞ . Other permutations of the vertices give other equivalent shape parameters $z' = \frac{z-1}{z}$ and $z'' = \frac{1}{1-z}$ but the underlying tetrahedron is the same. The construction is well defined as isometries of \mathbb{H}^3 are determined by their action on three vertices of $\partial\mathbb{H}^3$.

Another way of characterizing the shape of an ideal hyperbolic tetrahedron is to consider its dihedral angles, i.e., the angle formed by two faces meeting on a common edge; see Figure 2 (middle). In an ideal tetrahedron, opposite angles are equal, and the sum of the six dihedral angles is 2π . We denote in the following the dihedral angles of a tetrahedron by a triplet (α, β, γ) with $(\alpha + \beta + \gamma) = \pi$.

3 Angle structures and hyperbolic volume

In this section, we introduce notions connected to the computation of complete hyperbolic structures on triangulations, via optimization methods. The approach given in this section is another formulation of Thurston's gluing equations, which are non-linear in the complex shape parameters mentioned above, we refer the reader to [4, 15] for more details.

3.1 Linear equations and angle structures

Let T be a 1-vertex ideal triangulation of a knot complement M with n tetrahedra. Since opposite edges have the same dihedral angles, all possible shapes of the tetrahedra can be represented by a vertex in \mathbb{R}^{3n} . We define an *angle structure*:

► **Definition 2** (Angle structures). *Given an ideal triangulation T , an angle structure is a value assignment to the dihedral angles of the tetrahedra of T such that:*

1. *all the angles are strictly positive;*
2. *the three dihedral angles (α, β, γ) of a tetrahedron sum to π ;*
3. *the dihedral angles around each edge of T sum to 2π .*

The set of angle structures on T is denoted $\mathcal{A}(T)$.

Conditions 1 and 2 ensure the angles are in $(0, \pi)$ and that the tetrahedra have the same orientation. Condition 3 is necessary for points on the interior of edges to have a neighborhood isometric to a hyperbolic ball.

Constraints 1, 2, 3 are linear, hence the set $\mathcal{A}(T)$ of angle structures of a triangulation is the relative interior of a polytope in \mathbb{R}^{3n} . It is of dimension $n + |\partial M|$ where $|\partial M|$ is the number of cusps of M ; in the case of knot complements, there is a single cusp. This polytope satisfies:

► **Theorem 3** (Casson; see [8]). *Let T be an ideal triangulation of M , an orientable 3-manifold with toric cusps. If $\mathcal{A}(T) \neq \emptyset$, then M admits a complete hyperbolic metric.*

Furthermore, [4] describes precisely a generating family for the tangent space of $\mathcal{A}(T)$ that can be computed in polynomial time.

3.2 Maximizing the hyperbolic volume

The hyperbolic volume of an ideal hyperbolic tetrahedron with dihedral angles $(\alpha, \beta, \gamma) \in (0, \pi)^3$ is given by the function vol :

$$\text{vol}(\alpha, \beta, \gamma) = L(\alpha) + L(\beta) + L(\gamma)$$

where L is the Lobachevsky function $L(x) = -\int_0^x \log |2 \sin t| dt$. The volume functional can be extended to the whole polytope $\mathcal{A}(T)$ by summing the volumes of the hyperbolic tetrahedra. The following result is due independently to Casson and Rivin.

► **Theorem 4** (Casson, Rivin[14]). *Let T be an ideal triangulation with n tetrahedra of M , an orientable 3-manifold with boundary consisting of tori. Then a point $p \in \mathcal{A}(T) \subset \mathbb{R}^{3n}$ corresponds to a complete hyperbolic metric on the interior of M if and only if p is a critical point of the function vol .*

Additionally, the volume functional is concave on $\mathcal{A}(T)$ and the maximum can be computed via convex optimization methods.

We can finally define CHS that represent combinatorially complete hyperbolic metrics:

► **Definition 5** (Complete Hyperbolic Structure). *A Complete Hyperbolic Structure (CHS) is a triangulation T equipped with an angle structure corresponding to the maximum of vol over $\mathcal{A}(T)$.*

► **Remark 6.** Angle structures are not CHS as they do not prevent shearing singularities, see Figure 2 (right), and consequently may represent *non-complete* hyperbolic metrics.

4 Behavior of the optimization

Exploiting the formalism of the previous section, one can design an algorithm [4] to find and compute a CHS by first finding a point in the polytope $\mathcal{A}(T)$, then computing a basis of the tangent space of $\mathcal{A}(T)$, and then maximizing the (concave) hyperbolic volume functional on this subspace. If $\mathcal{A}(T) \neq \emptyset$ and the procedure finds the point maximizing the volume in the inside of the polytope, then the triangulation admits a complete hyperbolic structure represented by this point. Finally, if $\mathcal{A}(T) \neq \emptyset$ but the maximum of the volume functional is on the boundary, then one needs to re-triangulate the manifold in order to search for a triangulation admitting a CHS.

In this section, we study the outcome of the hyperbolic volume maximization on the space of angle structures. The result is given by the following lemma, which also appeared independently in [12]:

► **Lemma 7.** *Let T be an ideal triangulation of M , a non-compact orientable 3-manifold with toric cusps. Let p be the point maximizing vol over $\overline{\mathcal{A}(T)}$ (the topological closure of $\mathcal{A}(T)$), then at p , if a tetrahedron has an angle equal to 0, then all other angles of the tetrahedron are in $\{0, \pi\}$.*

In other words, either the maximization succeeds on the interior of $\mathcal{A}(T)$, or there is at least one tetrahedron with angles $(0, 0, \pi)$ in p . Such tetrahedron is called *flat*. It is not possible to have a tetrahedron with angles $(0, a, b)$, $(a, b) \in (0, \pi)$ in p .

This lemma is proven by looking at the derivative of the volume around the boundary of $\mathcal{A}(T)$, see [9].

We introduce in the next section an algorithm that performs localized combinatorial modifications on a triangulation equipped with an angle structure, in order to get rid of flat tetrahedra.

5 Localized combinatorial modifications of triangulations

According to Lemma 7, the volume maximization either leads to a solution to the gluing equations, or to flat tetrahedra. In this section, we discuss a method to get rid of flat tetrahedra by combinatorial modifications of the triangulation, while attempting to maintain the value of the volume functional. This would lead to a triangulation admitting an angle structure with larger volume than the previous one, allowing to resume the maximization. In order to maintain the value of the volume functional, we introduce *geometric Pachner moves*.

5.1 Geometric Pachner moves

We perform Pachner moves on the triangulation that preserves the partial geometric data computed. More precisely, we define:

► **Definition 8** (Geometric Pachner move). *A geometric Pachner move in a triangulation T with angle structure is a Pachner move in T such that the resulting triangulation admits an angle structure with identical dihedral angles for the tetrahedra not involved in the move.*

To check if a geometric Pachner move can be done, it must be valid combinatorially, and it should be possible to assign dihedral angles to the new tetrahedra without altering the rest of the angle structure. The combinatorial conditions are simple [13]: for the 2-3 moves (resp. 3-2 move) in Figure 3, the tetrahedra sharing the common triangle BDC (resp. common edge AE) must be distinct. The conditions on the angle structures are given by the following lemma:

► **Lemma 9.** *Given a triangulation with angle structure T , if the combinatorial conditions given above are satisfied:*

- *a 3-2 move is always geometric;*
- *a 2-3 move is geometric if and only if the sum of the two dihedral angles around each edge of the common face is smaller than π .*

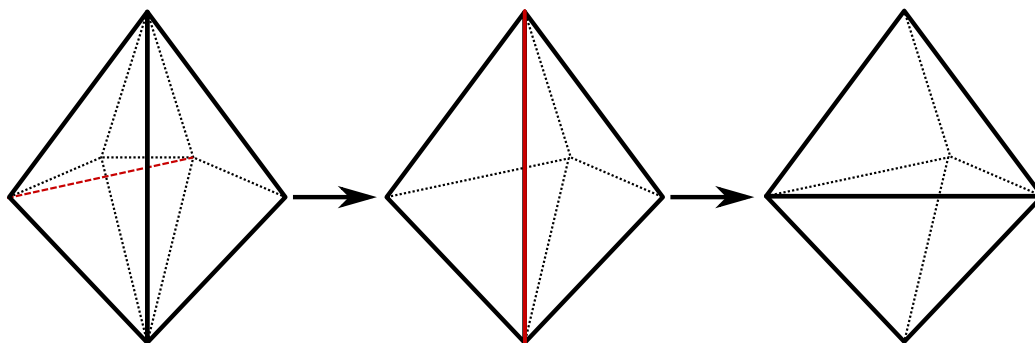
The idea to prove the lemma is to study the system of equations given by the assumption that the angle structure is not altered outside the move. Furthermore there are formulas to compute the new angles. The proofs of Lemma 9 and of this later claim are technical and can be found in [9].

► **Remark 10 (Non-conservation of the volume).** In Lemma 9, several angle structures are possible after the 2-3 move. Conversely, several angle structures lead to the same result for the 3-2 move. This is because angle structures are blind to shearing singularities, see Figure 2 (right). If such a singularity exists, a 3-2 move can still be performed while maintaining the whole angle structure, but it will decrease the hyperbolic volume of the structure. It is due to the fact that the singularity, which maximized the volume, is fixed in the process.

5.2 Getting rid of flat tetrahedra

Let T be a triangulation with an angle structure admitting a flat tetrahedron t , and (e, e') the associated edges with dihedral angle π . By Lemma 9, it is not possible to get rid of a tetrahedron with a geometric 2-3 move. Indeed, if an edge of t has a dihedral angle equal to π , the second tetrahedron concerned by the 2-3 move must have a dihedral angle equal to 0, and consequently a 2-3 move will produce a flat tetrahedron (see [9]).

In order to get rid of t , our strategy is to turn either e or e' into an edge of degree three, at the center of three tetrahedra on which a 3-2 move can be performed (central edge of Figure 3, right). Note that edges of degree 2 prevent the existence of an angle structure



■ **Figure 4** A sequence of moves getting rid of a flat tetrahedron. The bold vertical edge is contained in 4 tetrahedra, three of which are represented behind the edge. The fourth tetrahedron is implicit, situated at the front, and flat. First, create the red edge of the first drawing with a 2-3 move. Then delete the red edge of the second drawing with a 3-2 move.

as they force the two tetrahedra sharing the edge to be flat. As a consequence, we assume these configurations are removed, which can be done by `SnapPy`'s simplification for instance, see Section 6. Consequently, all edges have degree at least 3, and our strategy focuses on *reducing* the incidence degree of angle π edges.

To reduce the degree of an edge e , the strategy is to perform 2-3 moves on the tetrahedra containing e , such that each move reduces the degree of the edge by one. Either for topological (tetrahedron glued to itself) or geometrical reasons, these moves will not always be possible, and the order in which they are done matters. This is described in Figure 4.

► **Remark 11.** Doing a 2-3 move does not always reduce the degree of the edge, e.g., when a tetrahedron is represented several times around an edge. However, doing a move that does not decrease the degree of the edge may delete the multiple occurrences of a tetrahedron around the edge and allow to continue with the simplification.

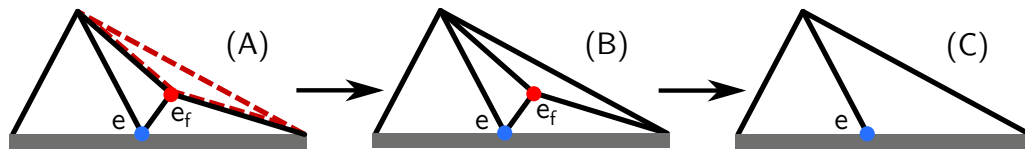
► **Remark 12.** When a 2-3 move is performed around e , the value of the dihedral angle of the new tetrahedron around e is equal to the sum of the previous two dihedral angles around e . Since the sum of the dihedral angles of a tetrahedron is equal to π , this means successfully reducing the degree of e may unlock previously forbidden moves.

Recursive moves. If no geometric Pachner move is possible on an edge e , we attempt to reduce the degree of a nearby edge e_f , i.e., an edge for which there exists a tetrahedron containing e and e_f ; see Figure 5 where e_f is represented by the red dot, as it is seen from above. More precisely, let f be a triangle containing e such that the associated 2-3 move is forbidden for geometric reasons, then by Lemma 9 there is an edge e_f of f for which the associated dihedral angle is at least π .

In consequence, reducing the degree of e_f to three without modifying the tetrahedra containing f and then performing a 3-2 move at e_f will reduce the degree of e by one; see Figure 5.

Note that in case a geometric Pachner move is not possible on any pair of tetrahedra containing e_f either, we can call the procedure recursively in a neighborhood of e_f , using the argument above to pass from e to e_f .

► **Remark 13.** The choice of e_f in f is unique: it is not possible to have two dihedral angles larger than π ; selecting a move to reduce the degree of e boils down to selecting a triangle containing e .



■ **Figure 5** Sequences of moves to reduce the degree of an edge when no 2-3 move is available, seen from above (triangles represent sections of tetrahedra, edge section of triangles and dots section of edges, see Figure 2 (Right)). In all drawings, the gray rectangle represents the flat tetrahedron, e (light blue dot) is the edge of the flat tetrahedron of which we want to reduce the degree. Assuming no 2-3 move is available, we pick another edge e_f (red dot), if a 3-2 move can be performed on e_f , we are in case (B) and the 3-2 move reduces the degree of e (case (C)). Otherwise, we are in case (A) we need to create the red dashed tetrahedron by decreasing the degree of e_f , this can be done by calling recursively our procedure on e_f .

Procedure summary. The procedure to get rid of a flat tetrahedron is a tree-like *backtracking search*, each branch corresponding to a choice of 2-3 or 3-2 move to perform. The moves used are geometric to preserve the volume maximization advancement. A flat tetrahedron can only be deleted with a 3-2 move: the first step is to *reduce the degree* of one of its edges to three. To reduce the degree of an edge, the solution is to perform 2-3 moves on two tetrahedra containing this edge. If a 2-3 move is not possible for geometric reasons, we are in the situation of Figure 5, and the procedure to *reduce the degree* can be called *recursively* on a neighboring edge. Performing a 3-2 move on this latter edge resulting in a decrease of the degree of the initial one. The procedure ends when the initial edge has degree three and the flat tetrahedron can be removed or when no move can be applied.

5.3 Implementation details

The implementation faces several practical challenges.

Breaking infinite loops. As such, the algorithm may loop infinitely on some instance, as 2-3 and 3-2 moves may reverse themselves. To counteract this phenomenon and avoid redundant modifications, a solution is to store at each modification the *isomorphism signature* of the triangulation [2], characterizing uniquely the isomorphism type of the triangulation. This allows us to recognize already processed triangulations and break branches of the backtracking algorithms. Note however that this is costly compared to the Pachner moves, and it makes the procedure no longer local.

Selecting the edge e . When attempting to remove a flat tetrahedron, one needs to choose between the two π -angled edges e and e' to reduce. In our implementation, we consider both edges, and select the one that has the larger smallest angle in its link. This performs better than a random choice in our experiments.

Pruning the backtrack search. Some triangulations may have edges of large degree (more than 10), which produces wide search trees. Additionally, the recursive calls to the degree reduction procedure may induce trees of large depth. Experimentally, the better strategy consists of exploring exhaustively the first levels of the tree. We set the width of exploration to 8 and explore the first 2 levels of the search tree.

Sequencing of Pachner moves. Different sequencings of Pachner moves lead to different triangulations. In practice, we favor 2-3 moves over recursive ones, which performs best. Additionally, among the 2-3 moves performed to get rid of a flat tetrahedron containing edge e of angle π , we prioritize moves that eliminate tetrahedra for which e has a small dihedral angle. Among the recursive ones, we used the same method as the one choosing between e and e' , indeed choosing a recursive move boils down to choosing an edge to minimize its degree.

6 Experiments

In this section, we study the experimental performance of our approach¹ to find a triangulation with a CHS, and compare its behavior with the software `SnapPy` [3]. `SnapPy` is the state-of-the-art software to study the geometric properties of knots and 3-manifolds, and is widely used in the low-dimensional topology community. Following this analysis, we propose and study a hybrid method with practical interest.

SnapPy. `SnapPy`'s method is based on a random re-triangulation followed by a simplification. The first step is constituted of $4n$ random 2-3 moves, where n is the number of tetrahedra in the input triangulation. The simplification performs non-deterministic modifications to decrease the number of tetrahedra in the triangulation. Notably it removes some configurations that prevent angle structures from existing. The verification of the existence of a CHS is based on a Newton's method to solve the gluing equations [17].

Data set. We apply our algorithms to the census of prime hyperbolic knots with up to 19 crossings. This census has been constituted by the efforts of many researchers in the field, and recently completed with the exhaustive enumeration of all knots with crossing number smaller than 20^2 [1].

For each knot, given by a knot diagram, we compute a triangulation of the knot complement using `Regina` [2], and simplify it with `SnapPy`. We then keep the triangulations admitting an angle structure but not a CHS, and hence requiring re-triangulation. All the others, without angle structure or admitting a CHS, were discarded. They constitute the *Failure on first try* data of Table 1.

The knots are grouped by crossing numbers (from 14 to 19), and on whether they are alternating or not. In Section 6.1, our experiments are run on the first 2500 knots of each of the 12 groups having an angle structure but no CHS, and in Section 6.2 we use the first 10.000 knots of the same data sets.

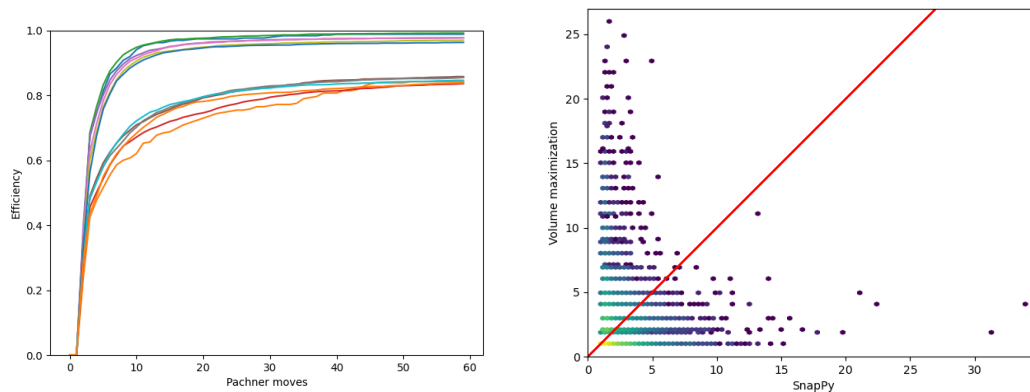
6.1 Success rate and combinatorial performance

In terms of Pachner moves. Figure 6 (left) represents the rate of knots on which our algorithm succeeds to find a triangulation with a CHS, for all the 12 groups of knots. It represents the success rate as a function of the number of Pachner moves, the higher success rate groups are the alternating knots.

While there is a significant difference between the efficiency on alternating and non-alternating knots, all the curves have the same behavior: the first few Pachner moves are very effective. An important point is that, compared to the number of moves done by `SnapPy`,

¹ Our implementation uses the SLSQP [7] optimization method from SciPy [16].

² The census is available at <https://regina-normal.github.io/data.html>



■ **Figure 6** Left: on the sample of triangulations, success rate in finding a CHS against the number of Pachner moves for different crossing numbers and alternabilities. The limit rate of success for alternating knots is 0.98, the limit rate for non-alternating knots is 0.87. Right: Number of re-triangulations required by our method over the number of re-triangulations required by SnapPy on the 12 groups of knots, gathered in bins (the color indicates the log of the number of points inside). Since our method is deterministic, the points have discrete ordinates. The diagonal $x = y$ appears in red.

at least four times the number of tetrahedra and then roughly the same number for the simplification times the number of re-triangulation, our method uses a much lower number of Pachner moves on average (see Figure 6, left). This is of *combinatorial interest*, as doing a small number of moves can mitigate the impact of the procedure on the properties of the triangulation, such as keeping a small treewidth for instance.

After the first few steps, the growth of the success rate slows down drastically. An interpretation of this phenomenon is that the tetrahedra created by the 2-3 moves tend to be more flat than the original ones. This leads to an increase in the number of required re-triangulations and issues with floating point arithmetic.

In terms of re-triangulation. In order to compare the behaviors of our method and of the state of the art: we want to know if the two methods struggle on the same triangulations. Both methods using sequences of re-triangulations, we use the length of these sequences as metrics.

We compare in Figure 6 (right) the average number of randomized re-triangulations required by SnapPy to find a triangulation with a CHS, and the (deterministic) number of re-triangulations of our method. The majority of the CHS are found within few re-triangulations on average: both methods manage to find 78% of them within two steps.

However, it appears that for the manifolds requiring a large number of re-triangulations with either method, the other one will perform well on it: the performance of the methods are substantially orthogonal for the difficult cases. This phenomenon highlights the fact that the methods are *complementary*.

Limitations. It is to be noted that our method suffers from several phenomena: there are manifolds on which our method fails, the reasons to this being convergence problems because of the floating point arithmetic, the creation of flatter tetrahedra or the volume modification mentioned in Remark 10. Furthermore some triangulations have several very high degree edges and looking for the correct sequence of geometric Pachner moves can be costly. In terms of time performance, the constraint convex optimization is an important bottleneck, as well as the computation and book-keeping of a large number of isomorphism signatures.

However, in light of the previous results, we design a hybrid algorithm, mixing `SnapPy`'s pipeline together with the heuristics based on localized geometric Pachner moves introduced in this article, in order to outperform both methods in terms of experimental timings.

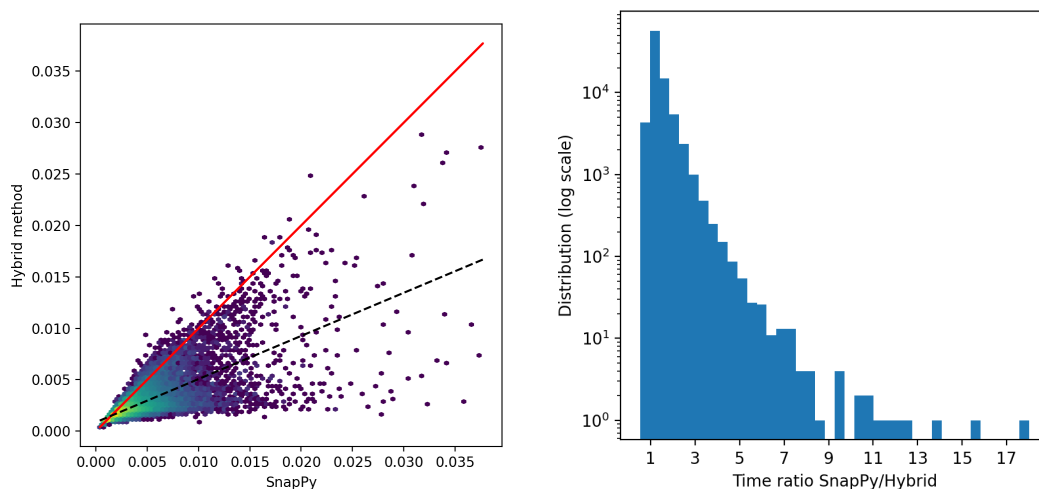
6.2 Hybrid algorithm and time performance

As is, `SnapPy` does not produce angle structures as defined in Section 3, as it constructs *negative tetrahedra*, i.e., tetrahedra whose shape has a negative imaginary part. Our hybrid method consists of calling `SnapPy` to randomize and simplify the triangulation, and, in case the triangulation admits only few negative tetrahedra (less than four in our experiments), call the resolution of flat tetrahedra with geometric Pachner moves on these negative tetrahedra, as introduced in Section 5. Loop until a CHS is found.

Our implementation is done in C directly in `SnapPy`'s kernel and consists of a simplified version of Section 5's method. For each negative tetrahedron, we try to reduce the degree of one of its edges down to 3 in turn using only geometric 2-3 moves, we try all the edges, and we do not use recursive moves or heuristics to perform the geometric moves.

The results of this method are summarized in Figure 7, where the time to compute the CHS are compared for the new hybrid method and the usual `SnapPy` pipeline, on the first 10.000 triangulations of the datasets of Section 6. The hybrid method using localized geometric Pachner moves shows, at worst, similar time performance compared to `SnapPy`, and performs significantly better overall. On average, the hybrid method is 20% faster, however, it is to be noted that, on most cases, few re-triangulations are required in which case the hybrid method and `SnapPy` show naturally similar time performance. More interestingly, on cases requiring many more re-triangulations in the `SnapPy` pipeline, the hybrid methods performs much better than `SnapPy`, with running times up to 18 times faster.

As indicator of the global behavior of both algorithms, we indicate the linear regression of the data points (purple dashed line, slope = 0.4) in Figure 7, to illustrate that the dense region is not concentrated on the $x = y$ diagonal.



■ **Figure 7** Left: time comparison of the hybrid method and `SnapPy` in seconds for the 10.000 first triangulations of each crossing number of the data set of the previous section, gathered in bins (the color indicates the log of the number of points inside). Line $x = y$ is plain and red, a linear regression through the data set is dashed and black. For readability, four outliers favoring the hybrid method are omitted. Right: distribution, in logarithmic scale, of the time required by `SnapPy` over the time required by the hybrid method.

References

- 1 Benjamin A. Burton. The Next 350 Million Knots. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2020.25.
- 2 Benjamin A. Burton, Ryan Budney, William Pettersson, et al. Regina: Software for low-dimensional topology. <http://regina-normal.github.io/>, 1999–2021.
- 3 Marc Culler, Nathan M. Dunfield, and Jeffrey R. Weeks. SnapPy, a computer program for studying the geometry and topology of 3-manifolds. <https://snappy.math.uic.edu/>, 1991–2021.
- 4 David Futer and François Guéritaud. From angled triangulations to hyperbolic structures, 2010. doi:10.1090/conm/541/10683.
- 5 Joel Hass, Jeffrey C. Lagarias, and Nicholas Pippenger. The computational complexity of knot and link problems. *J. ACM*, 46(2):185–211, 1999. doi:10.1145/301970.301971.
- 6 William H. Jaco and J. Hyam Rubinstein. 0-efficient triangulations of 3-manifolds. *Journal of Differential Geometry*, 65:61–168, 2002.
- 7 D. Kraft. *A Software Package for Sequential Quadratic Programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988.
- 8 Marc Lackenby. Word hyperbolic dehn surgery. *Inventiones mathematicae*, pages 243–282, 2000. doi:10.1007/s002220000047.
- 9 Clément Maria and Owen Rouillé. Computing complete hyperbolic structures on cusped 3-manifolds, 2021. doi:10.48550/ARXIV.2112.06360.
- 10 George D. Mostow. *Strong Rigidity of Locally Symmetric Spaces*. Princeton University Press, 1973. doi:doi:10.1515/9781400881833.
- 11 Hitoshi Murakami and Yoshiyuki Yokota. *Volume Conjecture for Knots*. Springer, 2018. doi:10.1007/978-981-13-1150-5.
- 12 Barbara Nimershiem. Geometric triangulations of a family of hyperbolic 3-braids, 2021. arXiv:2108.09349.
- 13 Udo Pachner. P.l. homeomorphic manifolds are equivalent by elementary shellings. *European Journal of Combinatorics*, 12(2):129–145, 1991. doi:10.1016/S0195-6698(13)80080-7.
- 14 Igor Rivin. Euclidean structures on simplicial surfaces and hyperbolic volume. *Annals of Mathematics*, 139:553–580, 1994.
- 15 W. P. Thurston. *The geometry and topology of 3-manifolds*, volume 1. Princeton University Press, Princeton, N.J., 1980. Electronic version 1.1 - March 2002. URL: <http://library.msri.org/books/gt3m/>.
- 16 Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi:10.1038/s41592-019-0686-2.
- 17 Jeffrey Weeks. Computation of hyperbolic structures in knot theory. *Handbook of Knot Theory*, October 2003. doi:10.1016/B978-044451452-3/50011-3.

Computing Treedepth in Polynomial Space and Linear FPT Time

Wojciech Nadara ✉

Institute of Informatics, University of Warsaw, Poland

Michał Pilipczuk ✉

Institute of Informatics, University of Warsaw, Poland

Marcin Smulewicz ✉

Institute of Informatics, University of Warsaw, Poland

Abstract

The *treedepth* of a graph G is the least possible depth of an *elimination forest* of G : a rooted forest on the same vertex set where every pair of vertices adjacent in G is bound by the ancestor/descendant relation. We propose an algorithm that given a graph G and an integer d , either finds an elimination forest of G of depth at most d or concludes that no such forest exists; thus the algorithm decides whether the treedepth of G is at most d . The running time is $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ and the space usage is polynomial in n . Further, by allowing randomization, the time and space complexities can be improved to $2^{\mathcal{O}(d^2)} \cdot n$ and $d^{\mathcal{O}(1)} \cdot n$, respectively. This improves upon the algorithm of Reidl et al. [ICALP 2014], which also has time complexity $2^{\mathcal{O}(d^2)} \cdot n$, but uses exponential space.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Graph algorithms analysis

Keywords and phrases treedepth, FPT, polynomial space

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.79

Related Version *Full Version*: <https://arxiv.org/abs/2205.02656> [13]

Funding *Wojciech Nadara*: This paper is a part of project CUTACOMBS that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 714704).

Michał Pilipczuk: This paper is a part of project BOBR that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 948057).

Acknowledgements We would like to thank Marcin Mucha and Marcin Pilipczuk for discussions on the topic of this work.

1 Introduction

An *elimination forest* of a graph G is a rooted forest F whose vertex set is the same as that of G , where for every edge uv of G , either u is an ancestor of v in F or vice versa. The *treedepth* of G is the least possible depth of an elimination forest of G . Compared to the better-known parameter *treewidth*, treedepth measures the depth of a tree-like decomposition of a graph, instead of width. The two parameters are related: if by $\text{td}(G)$ and $\text{tw}(G)$ we denote the treedepth and the treewidth of an n -vertex graph G , then it always holds that $\text{tw}(G) \leq \text{td}(G) \leq \text{tw}(G) \cdot \log_2 n$ (Section 6.4 of [16]). However, the two notions are qualitatively different: for instance, a path on t vertices has treewidth 1 and treedepth $\Theta(\log t)$.

Treedepth appears prominently in structural graph theory, especially in the theory of sparse graphs of Nešetřil and Ossona de Mendez. There, it serves as a basic building block for fundamental decompositions of sparse graphs – *low treedepth colorings* – which can



© Wojciech Nadara, Michał Pilipczuk, and Marcin Smulewicz;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, 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



be used for multiple algorithmic purposes, including designing algorithms for SUBGRAPH ISOMORPHISM and model-checking First-Order logic. See [16, Chapters 6 and 7] for an introduction and [8, 9, 15, 17, 18, 19, 21, 20] for examples of applications.

In this work we are interested in using treedepth as a parameter for the design of fixed-parameter (FPT) algorithms. Clearly, every dynamic programming algorithm working on a tree decomposition of a graph can be adjusted to work also on an elimination forest, just because an elimination forest of depth d can be easily transformed into a tree decomposition of width $d - 1$. However, it has been observed in [7, 22, 10, 14, 20] that for multiple basic problems, one can design FPT algorithms working on elimination forests of bounded depth that have polynomial space complexity without sacrificing on the time complexity. These include the following: (In all results below, n is the vertex count and d is the depth of the given elimination forest.)

- A $3^d \cdot n^{\mathcal{O}(1)}$ -time $\mathcal{O}(d + \log n)$ -space algorithm for 3-COLORING [22].
- A $2^d \cdot n^{\mathcal{O}(1)}$ -time $n^{\mathcal{O}(1)}$ -space algorithm for counting perfect matchings [7].
- A $3^d \cdot n^{\mathcal{O}(1)}$ -time $n^{\mathcal{O}(1)}$ -space algorithm for DOMINATING SET [7, 22].
- A $d^{|V(H)|} \cdot n^{\mathcal{O}(1)}$ -time $n^{\mathcal{O}(1)}$ -space algorithm for SUBGRAPH ISOMORPHISM [20].
(Here, H is the sought pattern graph.)
- A $3^d \cdot n^{\mathcal{O}(1)}$ -time $n^{\mathcal{O}(1)}$ -space algorithm for CONNECTED VERTEX COVER [10].
- A $5^d \cdot n^{\mathcal{O}(1)}$ -time $n^{\mathcal{O}(1)}$ -space algorithm for HAMILTONIAN CYCLE [14].

We note that the approach used in [10, 14] to obtain the last two results applies also to several other problems with connectivity constraints. However, as these algorithms are based on the Cut&Count technique [4], they are randomized and no derandomization preserving the polynomial space complexity is known. An in-depth complexity-theoretical analysis of the time-space tradeoffs for algorithms working on different graph decompositions can be found in [22].

In the algorithms mentioned above one assumes that the input graph is supplied with an elimination depth of depth at most d . Therefore, it is imperative to design algorithms that given the graph alone, computes, possibly approximately, such an elimination forest. Compared to the setting of treewidth and tree decompositions, where multiple approaches have been proposed over the years (see e.g. [2, 11] for an overview), so far there is only a handful of algorithms to compute the treedepth exactly or approximately.

- It is well-known (see e.g. [16, Section 6.2]) that just running depth-first search and outputting the forest of recursive calls gives an elimination forest of depth at most $2^{\text{td}(G)}$. So this gives a very simple linear-time approximation algorithm, but with the approximation factor exponential in the optimum.
- Czerwiński et al. [5] gave a polynomial-time algorithm that outputs an elimination forest of depth at most $\mathcal{O}(\text{td}(G)\text{tw}(G) \log^{3/2} \text{tw}(G))$, which is thus an $\mathcal{O}(\text{tw}(G) \log^{3/2} \text{tw}(G))$ -approximation algorithm.
- Reidl et al. [23] gave an exact FPT algorithm that in time $2^{\mathcal{O}(d \cdot \text{tw}(G))} \cdot n$ either constructs an elimination forest of depth at most d , or concludes that the treedepth is larger than d .
- There is a naive algorithm computing treedepth directly from its definition that works in $\mathcal{O}(n^{\text{td}(G)})$ time and uses polynomial space.
- Using a tradeoff that runs either of the last two approaches, depending on whether d is greater than $\sqrt{\epsilon \cdot \log n}$ or not, for any fixed $\epsilon > 0$, one can obtain an algorithm running in time $2^{\mathcal{O}(\frac{d^3}{\epsilon})} + \mathcal{O}(n^{1+\epsilon})$ and using polynomial space.

Recall here that $\text{tw}(G) \leq \text{td}(G)$, hence when parameterized by treedepth only, the mentioned results can be seen as an $\mathcal{O}(\text{td}(G) \log^{3/2} \text{td}(G))$ -approximation in polynomial time, as well as an exact FPT algorithm with running time $2^{\mathcal{O}(d^2)} \cdot n$. In particular, obtaining a constant-factor

approximation for treedepth running in time $2^{\mathcal{O}(\text{td}(G))} \cdot n^{\mathcal{O}(1)}$ is a well-known open problem, see e.g. [5]. We note that implementation of practical FPT algorithms for computing treedepth was the topic of the 2020 Parameterized Algorithms and Computational Experiments (PACE) Challenge [12].

Our contribution. The exact algorithm of Reidl et al. [23] uses not only exponential time (in the treedepth), but also exponential space. This would make it a space bottleneck when applied in combination with any of the polynomial-space algorithms developed in [7, 22, 10, 14, 20]. The mentioned tradeoff trick brings back the polynomial space, but significantly deteriorates the running time both in terms of the factor exponential in d and the one polynomial in n . In this work we bridge these issues by proving the following result.

► **Theorem 1.** *There is an algorithm that given an n -vertex graph G and an integer d , either constructs an elimination forest of G of depth at most d , or concludes that the treedepth of G is larger than d . The algorithm runs in $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ time and uses $n^{\mathcal{O}(1)}$ space.*

The space and time complexities can be improved to $d^{\mathcal{O}(1)} \cdot n$ and expected $2^{\mathcal{O}(d^2)} \cdot n$, respectively, at the cost of allowing randomization: the algorithm may return a false negative with probability at most $\frac{1}{c \cdot n^c}$, where c is any constant fixed a priori; there are no false positives.

Thus, the randomized variant of the algorithm of Theorem 1 has the same time complexity as the algorithm of Reidl et al. [23], but uses polynomial space. However, the algorithm of Reidl et al. [23] is deterministic, contrary to ours. Note that apart from possible false negatives, the bound on the running time is only in expectation and not worst-case (in other words, our algorithm is both Monte Carlo and Las Vegas). However, one can turn this into a worst-case bound at the cost of increasing the probability of false negatives to $1/2$ by forcefully terminating the execution if the algorithm runs for twice as long as expected.

Simultaneously achieving time complexity linear in n and polynomial space complexity is a property that is desired from an algorithm for computing the treedepth of a graph. While many of the polynomial-space FPT algorithms working on elimination forests do not have time complexity linear in n due to the usage of various algebraic techniques, the simplest ones that exploit only recursion – like the ones for 3-COLORING or INDEPENDENT SET considered in [22] – can be easily implemented to run in time $2^{\mathcal{O}(d)} \cdot n$ and space $d^{\mathcal{O}(1)} \cdot n$. Thus, the randomized variant of the algorithm of Theorem 1 would neither be a bottleneck from the point of view of space complexity nor from the point of view of the dependency of the running time on n . Admittedly, the parametric factor in the runtime of our algorithm is $2^{\mathcal{O}(d^2)}$, as compared to $2^{\mathcal{O}(d)}$ in most of the aforementioned polynomial-space FPT algorithms working on elimination forests; this brings us back to the open problem about constant-factor approximation for treedepth running in time $2^{\mathcal{O}(\text{td}(G))} \cdot n^{\mathcal{O}(1)}$ raised in [5].

Let us briefly discuss the techniques behind the proof of Theorem 1. The algorithm of Reidl et al. [23] starts by approximating the treewidth of the graph (which is upper bounded by the treedepth) and tries to construct an elimination forest of depth at most d by bottom-up dynamic programming on the obtained tree decomposition. By applying the iterative compression technique, we may instead assume that we are supplied with an elimination forest of depth at most $d + 1$, and the task is to construct one of depth at most d .

Applying now the approach of Reidl et al. [23] directly (that is, after a suitable adjustment from the setting of tree decompositions to the setting of elimination forests) would not give an algorithm with polynomial space complexity. The reason is that their dynamic programming procedure is quite involved and in particular keeps track of certain disjointness conditions; this is a feature that is notoriously difficult to achieve using only polynomial space. Therefore,

we resort to the technique of *inclusion-exclusion branching*, used in previous polynomial-space algorithms working on elimination forests; see [7, 22] for basic applications of this approach. In a nutshell, the idea is to count more general objects where the disjointness constraints are relaxed, and to use inclusion-exclusion at each step of the computation to make sure that objects not satisfying the constraints eventually cancel out. We note that while the application of inclusion-exclusion branching was rather simple in [7, 22], in our case it poses a considerable technical challenge. In particular, along the way we do not count single values, but rather polynomials with one formal variable that keeps track of how much the disjointness constraints are violated. In the exposition layer, our application of inclusion-exclusion branching mostly follows the algorithm for DOMINATING SET of Pilipeczuk and Wrochna [22].

In this way, we can count the number of elimination forests¹ of depth at most d in time $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ and using polynomial space. So in particular, we can decide whether there exists at least one such elimination forest. Such a decision algorithm can be quite easily turned into a construction algorithm using self-reducibility of the problem. This establishes the first part of Theorem 1.

As for the second part – the randomized linear-time FPT algorithm using polynomial space – there are several obstacles that need to be overcome. First, there is a multiplicative factor n in the running time coming from the iterative compression scheme. We mitigate this issue by replacing iterative compression with the recursive contraction scheme used by Bodlaender in his linear-time FPT algorithm to compute the treewidth of a graph [1]. Second, when using self-reducibility, we may apply the decision procedure n times, each taking at least linear time. This is replaced by an approach based on color coding, whose correctness relies on the fact that in a connected graph of treedepth at most d there are at most $d^{\mathcal{O}(d)}$ different feasible candidates for the root of an optimum-depth elimination tree [3]. Finally, in the counting procedure we may operate on numbers of bitsize as large as polynomial in n . This is resolved by hashing them modulo a random prime of magnitude $\Theta(\log n)$, so that we may assume that arithmetic operations take unit time.

We remark that it is relatively rare that a polynomial-space algorithm based on algebraic techniques can be also implemented so that it runs in time linear in the input size. Therefore, we find it interesting and somewhat surprising that this can be achieved for the problem of computing the treedepth of a graph, which combinatorially is rather involved.

Organization. After brief preliminaries in Section 2, in Section 3 we prove the first part of Theorem 1: we give a deterministic algorithm that runs in time $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ time and uses polynomial space. Then, in Section 4 we improve the time and space complexities to $2^{\mathcal{O}(d^2)}n$ and $d^{\mathcal{O}(1)}n$ respectively, at the cost of introducing randomization. The proofs of the results marked \star are deferred to the full version of the work due to the space constraints.

2 Preliminaries

Standard notation. All graphs in this paper are finite, undirected, and simple (i.e. with no loops on vertices or multiple edges with the same endpoints). For a graph G and a vertex subset $A \subseteq V(G)$, by $N_G[A]$ we denote the *closed neighborhood* of A : the set consisting of all vertices that are in A or have a neighbor in A .

¹ Formally, we count only elimination forests satisfying some basic connectivity property, which we call *sensibility*.

For a function $f: A \rightarrow B$ and a subset of the domain $X \subseteq A$, by $f(X)$ we denote the image of f on X . The image of f is denoted $\text{im}(f) = f(A)$. For an element e outside of the domain and a value α , by $f[e \rightarrow \alpha]$ we denote the extension of f obtained by additionally mapping e to α .

We denote the set $\{1, 2, \dots, k\}$ as $[k]$. We assume the standard word RAM model of computation with words of length $\log n$, where n is the vertex count of the input graph.

(Elimination) forests and treedepth. Consider a rooted forest F . By Anc_F we denote the ancestor/descendant relation in F : for $u, v \in V(F)$, $\text{Anc}_F(u, v)$ holds if and only if u is an ancestor of v or v is an ancestor of u in F . We assume that a vertex is an ancestor of itself, so in particular $\text{Anc}_F(u, u)$ is always true. We also use the following notation. For $u \in V(F)$, by $\text{tail}_F[u]$ we denote the set of all ancestors of u (including u) and by $\text{tree}_F[u]$ we denote the set of all descendants of u , including u . Further, let $\text{tail}_F(u) = \text{tail}_F[u] - \{u\}$, $\text{tree}_F(u) = \text{tree}_F[u] - \{u\}$, and $\text{comp}_F[u] = \text{tail}_F[u] \cup \text{tree}_F[u]$. Note that $v \in \text{comp}_F[u]$ if and only if $\text{Anc}_F(u, v)$ holds. By $\text{chld}_F(u)$ we denote the set of children of u in F , and by $\text{depth}_F(u)$ we denote the depth of u in F , that is, $\text{depth}_F(u) = |\text{tail}_F[u]|$ (in particular, roots have depth one). The *depth* of a rooted forest F is the maximum depth_F among its vertices. For a set of vertices $A \subseteq V(F)$, by $\text{cl}_F(A) = \bigcup_{u \in A} \text{tail}_F[u]$ we denote the ancestor closure of A . A *prefix* of a rooted forest F is a rooted forest induced by some ancestor-closed set $A \subseteq V(F)$; that is, it is the forest on A with the parent-child relation inherited from F .

In this paper we are mostly interested in the notion of an elimination forest and of the treedepth of a graph.

► **Definition 2.** An elimination forest of a graph G is a rooted forest F on the same set of vertices as G such that for every edge $uv \in E(G)$, we have that $\text{Anc}_F(u, v)$ holds. The treedepth of a graph G is the least possible depth of an elimination forest of G .

Note that an elimination forest of a connected graph must be connected as well, so in this case we may speak about an *elimination tree* (however, elimination forest of a disconnected graph could be a tree as well). Sometimes, instead of identifying $V(G)$ and $V(F)$, we treat them as disjoint sets and additionally provide a bijective mapping $\phi: V(G) \rightarrow V(F)$ such that $uv \in E(G)$ entails $\text{Anc}_F(\phi(u), \phi(v))$. In such case we consider the pair (F, ϕ) to be an elimination forest of G . This will be always clear from the context. More generally, for $B \subseteq V(G)$ and a rooted forest F , we shall say that a mapping $\phi: B \rightarrow V(F)$ respects edges if $uv \in E(G)$ entails $\text{Anc}_F(\phi(u), \phi(v))$ for all $u, v \in B$. In this notation, (F, ϕ) is an elimination forest of G if and only if ϕ is a bijection from $V(G)$ to $V(F)$ that respects edges on $V(G)$.

3 Deterministic FPT algorithm

In this section we prove the first part of Theorem 1: we give a deterministic polynomial-space algorithm with running time $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ that for a given n -vertex graph G , either outputs an elimination forest of G of depth at most d or concludes that no such forest exists. The most complex part of the algorithm will be procedure `CountElimTrees`, which, roughly speaking, counts the number of different elimination trees of a connected graph G of depth at most d . We describe `CountElimTrees` first, and then we utilize it to achieve the main result.

3.1 Description of `CountElimTrees`

As mentioned above, procedure `CountElimTrees` counts the number of different elimination trees of G of depth at most d . However, we will not count all of them, but only such that are in some sense minimal; a precise formulation will follow later. We remark that this part is

inspired by the $3^d \cdot n^{\mathcal{O}(1)}$ -time polynomial space algorithm of Pilipczuk and Wrochna [22] for counting dominating sets in a graph of bounded treedepth. This algorithm exploits the same underlying trick – sometimes dubbed “inclusion-exclusion branching” – but the application here is technically more involved than in [22].

Before describing `CountElimTrees`, let us carefully define objects that we are going to count. We start by recalling the following standard fact about the existence of elimination forests with basic connectivity properties.

► **Lemma 3 (★)**. *Let H be a graph and let R be an elimination forest of H . Then there exists an elimination forest R' of H such that*

- *for every vertex u of H , we have $\text{depth}_{R'}(u) \leq \text{depth}_R(u)$; and*
- *whenever vertices $u, v \in V(H)$ belong to the same connected component of R' , they also belong to the same connected component of H .*

We remark that computing R' can be easily done in linear time by using depth-first search from the root of each elimination tree in R . This procedure will be used many times throughout the algorithm when justifying the usual assumption that our current graph is connected. (Disconnected graphs will often naturally appear when recursing after performing some deletions in the original graph.)

The following lemma can be proved using a very similar, though a bit more involved reasoning. We will work with a fixed connected graph G and its elimination tree T .

► **Lemma 4 (★)**. *Let G be a connected graph of treedepth at most d and T be an elimination tree of G (possibly of depth larger than d). Then there exists an elimination tree R of G of depth at most d that satisfies the following property: for every $u \in V(G)$ and $v_1, v_2 \in \text{chld}_T(u)$, $v_1 \neq v_2$, we have*

$$\text{cl}_R(\text{comp}_T[v_1]) \cap \text{cl}_R(\text{comp}_T[v_2]) = \text{cl}_R(\text{tail}_T[u]). \quad (1)$$

An elimination tree R of a graph G satisfying the conclusion of Lemma 4 (that is, the depth of R is at most d and for all $u \in V(G)$ and distinct $v_1, v_2 \in \text{chld}_T(u)$ we have (1)) will be called *sensible* with respect to T . In our search for elimination trees of low depth, we will restrict attention only to trees that are sensible with respect to some fixed elimination tree T . Then Lemma 4 justifies that we may do this without losing all solutions.

With all ingredients introduced, we may finally precisely state the goal of this section.

► **Lemma 5**. *There exists an algorithm `CountElimTrees`(G, T, d) that, given a connected graph G on n vertices, an elimination tree T of depth k , and an integer d , runs in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$, uses $n^{\mathcal{O}(1)}$ space, and outputs the number of different elimination trees of G of depth at most d that are sensible with respect to T .*

Note here that the input to `CountElimTrees` consists not only of G and d , but also of an auxiliary elimination tree T of G . The depth k of T may be, and typically will be, larger than d . Also, we assume that an elimination tree is represented solely by its vertex set and the ancestor relation. In particular, permuting children of a vertex yields the *same* elimination tree, which should be counted as the same object by procedure `CountElimTrees`.

The remainder of this section is devoted to the proof of Lemma 5. We first need to introduce some definition.

Let us arbitrarily enumerate the vertices of G as v_1, v_2, \dots, v_n in a top-down manner in T . That is, whenever v_i is an ancestor of v_j , we have $i \leq j$. Consider another rooted tree R and a mapping $\phi: V(T) \rightarrow V(R)$. For a vertex u of T , we call a vertex $v_i \in \text{tree}_T(u)$ a *proper surplus image* (for u and (R, ϕ)) if at least one of the following conditions holds:

- (i) $\phi(v_i) \in \text{cl}_R(\phi(\text{tail}_T[u]))$, or
- (ii) there exists j such that $j < i$, $v_j \in \text{tree}_T(u)$, and $\phi(v_j) = \phi(v_i)$.

We define *non-proper surplus images* analogously, but using sets $\text{tail}_T(u)$ and $\text{tree}_T[u]$ instead of $\text{tail}_T[u]$ and $\text{tree}_T(u)$, respectively.

We will work in the ring of polynomials $\mathbb{Z}[x]$, where x is a formal variable. By an abuse of notation, we equip this ring with an operation of division by x defined through equations:

$$\frac{x^i}{x} = \begin{cases} x^{i-1} & \text{if } i \geq 1, \\ 0 & \text{if } i = 0 \end{cases}$$

$$\frac{\alpha A + \beta B}{x} = \alpha \cdot \frac{A}{x} + \beta \cdot \frac{B}{x} \quad \text{for all } A, B \in \mathbb{Z}[x] \text{ and } \alpha, \beta \in \mathbb{Z}.$$

Formally speaking, division by x is just the unique function from $\mathbb{Z}[x]$ to $\mathbb{Z}[x]$ satisfying the two properties above.

Even though our final goal is to count the number of elimination trees, along the way we are going to count more general objects, called *generalized elimination trees*. A generalized elimination tree of a graph H is a rooted tree R along with a mapping $\phi: V(H) \rightarrow V(R)$ such that ϕ respects edges. Note that in particular, it may be the case that $\text{im}(\phi) \subsetneq V(R)$ or that $\phi(u) = \phi(v)$ for some $u, v \in V(H)$. Clearly, a generalized elimination tree is an elimination tree in the usual sense if and only if ϕ is a bijection between $V(H)$ and $V(R)$. We shall call two generalized elimination trees (R, ϕ) and (R', ϕ') *isomorphic* if there is an isomorphism of rooted trees ψ mapping R to R' such that $\phi' = \psi \circ \phi$.

A generalized elimination tree (R, ϕ) of an induced subgraph H of G is *sensible* for T if for every $u \in V(H)$ and distinct $v_1, v_2 \in \text{chld}_T(u) \cap V(H)$, we have $\text{cl}_R(\phi(\text{comp}_T[v_1])) \cap \text{cl}_R(\phi(\text{comp}_T[v_2])) = \text{cl}_R(\phi(\text{tail}_T[u]))$. Thus, this notion projects to sensibility of (standard) elimination trees when $H = G$ and (R, ϕ) is an elimination tree of G . Generalized elimination trees of induced subgraphs of G that are sensible for T shall be called *monsters*.

For a rooted tree K , a mapping ϕ with co-domain $V(K)$ is called a *cover* of K if $\text{cl}_K(\text{im}(\phi)) = V(K)$, or equivalently, every leaf of K is in the image of ϕ . For a vertex $u \in V(G)$, rooted tree K of depth at most d , a subset of vertices $A \subseteq V(K)$ that contains all leaves of K , and a mapping $\phi: \text{tail}_T(u) \rightarrow A$ that is a cover of K , we define

$$f(u, K, \phi, A) = \sum_{i=0}^n a_i x^i \in \mathbb{Z}[x],$$

where a_i is the number of non-isomorphic monsters $(R, \bar{\phi})$ such that:

- (i) $(R, \bar{\phi})$ is a generalized elimination tree of $G[\text{comp}_T[u]]$ of depth at most d ;
- (ii) K is a prefix of R ;
- (iii) $\bar{\phi}$ is an extension of ϕ satisfying

$$V(R) - V(K) \subseteq \text{im}(\bar{\phi}) \subseteq (V(R) - V(K)) \cup A; \quad \text{and}$$

- (iv) in $\text{tree}_T[u]$ there are exactly i non-proper surplus images for u and $(R, \bar{\phi})$.

Note that since ϕ is assumed to be a cover of K , and by the second and third condition, the last condition can be rephrased as follows:

$$i = |\text{tree}_T[u]| - |V(R) - V(K)|.$$

We define polynomial $g(u, K, \phi, L)$ analogously, but using $\text{tail}_T[u]$, $\text{tree}_T(u)$, and proper surplus images, instead of $\text{tail}_T(u)$, $\text{tree}_T[u]$ and non-proper surplus images. That in $\text{tree}_T(u)$ there are i proper surplus images is then equivalent to $i = |\text{tree}_T(u)| - |V(R) - V(K)|$.

Informally, f and g count partial solutions on subgraphs induced on subtrees of T , where in g we exclude the root u of the subtree. Values of g are computed by combining results of f from separate subtrees rooted at the children u into the result for the forest representing their union. Values of f are computed from the values of g by including a new vertex u that connects that forest into one tree.

Our goal now is to compute the polynomials $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$ recursively over the elimination tree T . It can be easily seen that if $\text{chld}_T(u) = \emptyset$ then

$$g(u, K, \phi, A) = \begin{cases} 1 & \text{if } \phi \text{ respects edges,} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Indeed, $(R, \bar{\phi}) = (K, \phi)$ is the only possible pair that can satisfy the last three conditions, and it is a sensible generalized elimination tree of $G[\text{comp}_T[u]]$ if and only if ϕ respects edges.

First, we show how to compute polynomials $g(u, \cdot, \cdot, \cdot)$ based on the knowledge of polynomials $f(v, \cdot, \cdot, \cdot)$ for children v of u .

► **Lemma 6 (★).** *If $\text{chld}_T(u) \neq \emptyset$, then for all relevant u, K, ϕ, A we have*

$$g(u, K, \phi, A) = \prod_{v \in \text{chld}_T(u)} f(v, K, \phi, A)$$

The detailed proof of this Lemma can be found in the full version, however a short justification is that monsters counted in the definition of $g(u, K, \Phi, A)$ can be expressed in a product structure of monsters counted in the definitions of $f(v_i, K, \Phi, A)$, where $\text{chld}_T(u) = \{v_1, \dots, v_c\}$.

Let us elaborate on the intuition on what happened in Lemma 4. Intuitively, we aggregated information about the children of u to the information about u itself. Since in the definitions of monsters we do not insist on the mappings being injective, this aggregation could have been performed by a simple product of polynomials (though, the assumption of sensibility was crucial for arguing the correctness). In a natural dynamic programming, such as the one in [23], one would need to ensure injectivity when aggregating information from the children of u , which would result in a dynamic programming procedure that would need to keep track of all subsets of K (and thus use exponential space). Thus, relaxing injectivity here allows us to use simple multiplication of polynomials, but obviously we will eventually need to enforce injectivity. The idea is that we enforce surjectivity instead, and make sure that the size of the co-domain matches the size of the domain. In turn, surjectivity is enforced using inclusion-exclusion in the computation of polynomials $f(u, \cdot, \cdot, \cdot)$ based on polynomials $g(u, \cdot, \cdot, \cdot)$, which is the subject of the next lemma. Ensuring that the size of the co-domain matches the size of the domain is done through maintaining the number of surplus images. The intuition behind their somewhat intricate definition is the following. We want to maintain the difference between (i) the number of vertices we have already processed and forgotten about, and (ii) the number of forgotten vertices in a monster that we introduced in order to accommodate the vertices (i). Vertices contributing to this difference are exactly the vertices that have been mapped to forgotten vertices of a monster that have already been “taken” or that have not been forgotten yet; this corresponds to the definition of surplus images. We know that if multiple vertices were mapped to the same vertex of a partial monster, then this partial monster will not become a valid treedepth decomposition. We do not have a way of discovering this immediately (as we cannot keep track of any disjointness conditions), but extensions of such partial monsters will not be counted in the final result. The reason for that is that either the sizes of the co-domains will not match the sizes of the domains, or they will cancel out in the exclusion-inclusion computation due to not being surjective.

► **Lemma 7 (★).** *For all relevant u, K, ϕ, A , we have:*

$$f(u, K, \phi, A) = \sum_{v \in A} x \cdot g(u, K, \phi[u \rightarrow v], A) + \sum_{w \in K} \sum_{p=1}^{d-\text{depth}(w)} \frac{1}{x^{p-1}} \sum_{B \subseteq \{w_1, \dots, w_{p-1}\}} (-1)^{p-1-|B|} g(u, K[w, w_1, \dots, w_p], \phi[u \rightarrow w_p], A \cup B \cup \{w_p\}),$$

where $K[w, w_1, \dots, w_p]$ denotes the rooted tree obtained from K by adding a path $[w, w_1, \dots, w_p]$ so that w is the parent of w_1 and each w_i is the parent of w_{i+1} , for $i \in \{1, \dots, p-1\}$.

We need to take an additional care of how to deduce the overall number of elimination trees based on the polynomial $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$. Define polynomial

$$h = \sum_{p=1}^d \frac{1}{x^{p-1}} \sum_{B \subseteq \{w_1, \dots, w_{p-1}\}} (-1)^{p-1-|B|} g(r, [w_1, \dots, w_p], [r \rightarrow w_p], B \cup \{w_p\}) \in \mathbb{Z}[x],$$

where r is the root of T , $[w_1, \dots, w_p]$ is a path on p vertices rooted at w_1 , and $[r \rightarrow w_p]$ denotes the function with domain $\{r\}$ that maps r to w_p .

► **Lemma 8.** *The number of elimination trees of G that are sensible with respect to T and have depth at most d is the term in h standing by x^0 .*

Proof. By Lemma 7, the formula can be seen as the formula for $f(r, K, \phi, A)$ for empty K , ϕ , and A . Therefore, h can be written as $h = \sum_{i=0}^n a_i x^i$, where a_i is the number of non-isomorphic sensible generalized elimination trees $(R, \bar{\phi})$ such that R has depth at most d , $\bar{\phi}: V(G) \rightarrow V(R)$ is surjective, and in G there are i non-proper surplus images for r and $(R, \bar{\phi})$. However, since K is empty, the number of surplus images is exactly the number of vertices $v_j \in V(G)$ that are mapped by $\bar{\phi}$ to the same vertex of R as some other vertex of G with a smaller index. Then the assertion that $\bar{\phi}$ is injective is equivalent to the assertion that the number of such surplus images is 0. It follows that the number of non-isomorphic sensible elimination trees of G of depth at most d is equal to the term in h that stands by x^0 . ◀

Having established Lemmas 6, 7 and 8, we can conclude the description of procedure `CountElimTrees`. By 8, the goal is to compute polynomial h and return the coefficient standing by x^0 . We initiate the computation using the formula for h , and then we use two mutually-recursive procedures to compute polynomials $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$ using formulas provided by Lemmas 6 and 7. The base case of recursion is for a leaf of T , where we use formula (2).

The correctness of the procedure is established by Lemmas 6, 7 and 8. So it remains to bound its time complexity and memory usage. It is clear that polynomials that we compute will always have degrees at most n . Trees K relevant in the computation will never have more than dk vertices, for at every recursive call the tree K can grow by at most d new vertices.

As the next step, we bound the numbers that can be present in the computations.

► **Lemma 9 (★).** *Every coefficient of $f(u, K, \phi, A)$ is an integer from the range $[0, (dk \cdot 2^d)^{|\text{tree}_T(u)|}]$ and every coefficient of $g(u, K, \phi, A)$ is an integer from the range $[0, (dk \cdot 2^d)^{|\text{tree}_T(u)|}]$. Hence, all integers present in the computations are at most $(dk2^d)^n$.*

79:10 Computing Treedepth in Polynomial Space and Linear FPT Time

It follows that all integers present in the computation have bitsize bounded polynomially in n .

As for the memory usage, the run of the algorithm is a recursion of depth bounded by $2k$. The memory used is a stack of at most $2k$ frames for recursive calls of procedures computing polynomials $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$ for relevant arguments. Each of these frames requires space polynomial in n , hence the total space complexity is polynomial in n .

As for the time complexity, each call to a procedure computing a polynomial of the form $f(u, \cdot, \cdot, \cdot)$ makes at most $dk \cdot 2^d$ recursive calls to procedures computing polynomials of the form $g(u, \cdot, \cdot, \cdot)$. In turn, each of these calls makes one call to a procedure computing a polynomial of the form $f(v, \cdot, \cdot, \cdot)$ for each child v of u . It follows that the total number of calls to procedures computing polynomials of the form $f(u, \cdot, \cdot, \cdot)$ and $g(u, \cdot, \cdot, \cdot)$ is bounded by $2 \cdot (dk \cdot 2^d)^k = 2^{\mathcal{O}(dk)}$. The internal work needed in each recursive call is bounded by $2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)}$. As T has n vertices, the total time complexity is $2^{\mathcal{O}(dk)} \cdot 2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)} \cdot n = 2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$, as claimed. This concludes the proof of Lemma 5.

We note that having designed $\text{CountElimTrees}(G, T, d)$, it is easy to design a similar function $\text{CountElimForest}(G, T, d)$ that does not need an assumption of G being connected and where T is some elimination forest instead of an elimination tree (by using the procedure described after Lemma 3).

3.2 Utilizing CountElimTrees

With the description of CountElimTrees completed, we can describe how we can utilize it in order to construct a bounded-depth elimination tree of a graph. That is, we prove the first part of Theorem 1.

First, we lift CountElimTrees to a constructive procedure that still requires to be provided an auxiliary elimination tree of the graph.

► **Lemma 10.** *There is an algorithm $\text{ConstructElimForest}(G, T, d)$ that, given an n -vertex graph G , an elimination forest T of G of depth at most k , and an integer d , runs in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$, uses $n^{\mathcal{O}(1)}$ space, and either correctly concludes that $\text{td}(G) > d$ or returns an elimination forest of G of depth at most d .*

Proof. By treating every connected component separately, we may assume that G is connected (see the remark after Lemma 3). Thus T is an elimination tree of G .

The first step of $\text{ConstructElimForest}(G, T, d)$ is calling $\text{CountElimTrees}(G, T, d)$. If this call returns 0, we terminate $\text{ConstructElimForest}$ and report that $\text{td}(G) > d$; this is correct by Lemma 4. Otherwise we are sure that $\text{td}(G) \leq d$, and we need to construct any elimination tree of depth at most d . In order to do so, we check, for every vertex $v \in V(G)$, whether v is a feasible candidate for the root of desired elimination tree. Note that a vertex v can be the root of an elimination tree of G of depth at most d if and only if $\text{td}(G - v) < d$, or equivalently, if and only if the procedure $\text{CountElimForest}(G - v, T - v, d - 1)$ returns a positive value. (Here, by $T - v$ we mean the forest T with v removed and all former children of v made into children of the parent of v , or to roots in case v was a root.) As $\text{td}(G) \leq d$, we know that for at least one vertex v , this check will return a positive outcome. Then we recursively call $\text{ConstructElimForest}(G - v, T - v, d - 1)$, thus obtaining an elimination forest F' of $G - v$ of depth at most $d - 1$, and we turn it into an elimination tree F of G by adding v as the new root and making it the parent of all the roots of F' . As F has depth at most d , it can be returned as the result of the procedure.

That the procedure is correct is clear. As for the time and space complexity, it is easy to see that there will be at most dn calls to the procedure `CountElimTrees` in total, because at each level of the recursion there will be at most one invocation of `CountElimTrees` per vertex of the original graph. As each of these calls uses $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ time and $n^{\mathcal{O}(1)}$ space, the same complexity bounds also follow for `ConstructElimForest`. ◀

It remains to show how to lift the assumption of being provided an auxiliary elimination forest of bounded depth. For this we use the iterative compression technique.

Proof of the first part of Theorem 1. Arbitrarily enumerate the vertices of G as v_1, \dots, v_n . For $i \in \{1, \dots, n\}$, let $G_i = G[\{v_1, \dots, v_i\}]$ be the graph induced by the first i vertices. For each $i = 1, 2, \dots, n$ we will compute F_i , an elimination forest of G_i of depth at most d . For $i = 1$ this is trivial. Assume now that we have already computed F_i and want to compute F_{i+1} . We first construct T_{i+1} , an elimination tree of G_{i+1} , by taking F_i , adding v_{i+1} , and making v_{i+1} the parent of all the roots of F_i . Note that T_{i+1} has depth at most $d + 1$. We now call `ConstructElimForest`(G_{i+1}, T_{i+1}, d). If this procedure concludes that $\text{td}(G_{i+1}) > d$, then this implies that $\text{td}(G) > d$ as well, and we can terminate the algorithm and provide a negative answer. Otherwise, the procedure returns an elimination forest F_{i+1} of G_{i+1} of depth at most d , with which we can proceed. Eventually, the algorithm constructs an elimination forest $F = F_n$ of $G = G_n$ of depth at most d .

The algorithm is clearly correct. Since every call to `ConstructElimForest` is supplied with an elimination forest of depth at most $d + 1$, and there are at most n calls, the total time complexity is $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ and the space complexity is $n^{\mathcal{O}(1)}$, as desired. ◀

4 Randomized linear FPT algorithm

In this section we sketch the second part of Theorem 1, where we reduce the time and space complexities to linear in n at the cost of relying on randomization. There are three main reasons why the algorithm presented in the previous section does not run in time linear in n .

- First, in procedure `ConstructElimForest`, we applied `CountElimTrees` $\mathcal{O}(dn)$ times. Even if `CountElimTrees` runs in time linear in n , this gives at least a quadratic time complexity for `ConstructElimForest`.
- Second, in the iterative compression scheme we add vertices one by one and apply procedure `ConstructElimForest` n times. Again, even if `ConstructElimForest` runs in linear time, this gives at least a quadratic time complexity.
- Third, in procedure `CountElimTrees` we handle polynomials of degree at most n and with coefficients of bitsize bounded only polynomially in n . Algebraic operations on those need time polynomial in n .

In short, the second and third obstacles are mitigated as follows:

- Iterative compression is replaced by a contraction scheme of Bodlaender [1] that allows us to replace iteration with recursion, where every recursive step reduces the total number of vertices by a constant fraction, rather than peels off just one vertex.
- We observe that in `CountElimTrees`, we may care only about monomials with degrees bounded by dk , so the degrees are not a problem. As for coefficients, we hash them modulo a sufficiently large prime. This is another source of randomization.

As the techniques discussed above are rather standard, their details can be found in the full version only. Here we provide an overview of how we optimize the procedure `ConstructElimForest`, as this part contains original ideas.

Faster root recovery

We assume we have already improved the running time of `CountElimTrees` to linear. Now we are going to improve the running time of `ConstructElimForest` to linear. Recall that `ConstructElimForest` in its current version works over a connected graph G , iterates over all vertices $v \in V(G)$ and checks whether $\text{td}(G - v) \leq d - 1$ by calling `CountElimTrees` with appropriate parameters. Such vertices v could be placed as roots of an elimination tree of G of depth at most d . Finding any feasible root is the crucial part that needs to be optimized in order to achieve a linear running time for `ConstructElimForest`. The key fact we are going to use is that the number of possible roots of optimum-depth elimination forests of a connected graph is bounded in terms of the treedepth [3, 6].

► **Lemma 11 (★).** *Suppose G is a graph whose treedepth is equal to d . Then there are at most $d^{\mathcal{O}(d)}$ vertices $v \in V(G)$ such that $\text{td}(G - v) < d$.*

Observe that supposing G is connected, vertices v satisfying $\text{td}(G - v) < \text{td}(G)$ are exactly those that can be placed as roots of an optimum-depth elimination tree.

We are going to modify the procedure `CountElimTrees`(G, T, d) by introducing weights. Let G be a connected graph. Enumerate vertices of G as $V(G) = \{v_1, \dots, v_n\}$ and let t_i be the number of elimination trees of G that are sensible with respect to T and in which v_i is the root. Then the result of `CountElimTrees`(G, T, d) can be expressed as $t_1 + t_2 + \dots + t_n$. However, with a slight modification, we are able to compute $t_1\mu_1 + t_2\mu_2 + \dots + t_n\mu_n$ for any sequence $\mu_1, \mu_2, \dots, \mu_n \in \mathbb{Z}$. That can be achieved by multiplying by μ_i the contribution of transitions when we map v_i to the root of a monster.

Assume wishfully that there is exactly one vertex $v_i \in V(G)$ that could serve as the root of an elimination tree of G of depth d ; equivalently, v_i is the only vertex such that $\text{td}(G - v_i) < d$. In other words, t_j is nonzero if and only if $i = j$. Note that in such case we have $i = \frac{\sum_{j=1}^n j \cdot t_j}{\sum_{j=1}^n t_j}$. The denominator of this expression is simply the number of all elimination trees of G of depth at most d that are sensible with respect to T , while the numerator is the result of the modified version of `CountElimTrees` where we set $\mu_j = j$ for all $j \in [n]$. Hence, we can find i (that is: pinpoint the unique root) by dividing the outcomes of two calls to weighted `CountElimTrees`, instead of calling `CountElimTrees` n times, as we did previously.

Next, we lift the assumption about the uniqueness of the candidate for the root of an elimination tree. There are two key ingredients here. The first one is Lemma 11, which bounds the number of possible candidate roots for elimination trees of optimum depth. The second one is the color coding technique. The idea is to randomly color vertices into $d^{\mathcal{O}(d)}$ colors. Because there are at most $d^{\mathcal{O}(d)}$ possible roots, with high probability there will exist a color such that there is exactly one possible root in it. By modifying the idea from the previous paragraph, we can generalize it to identifying root within a color class provided there is exactly one possible root of this color.

Similarly as in the slower version, after identifying any possible root, we can delete it and recurse to connected components of the remaining part of the graph.

References

- 1 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 2 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

- 3 Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski, and Anna Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 796–809. SIAM, 2021. doi:10.1137/1.9781611976465.50.
- 4 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.
- 5 Wojciech Czerwiński, Wojciech Nadara, and Marcin Pilipczuk. Improved bounds for the excluded-minor approximation of treedepth. *SIAM J. Discret. Math.*, 35(2):934–947, 2021. doi:10.1137/19M128819X.
- 6 Zdeněk Dvořák, Archontia C. Giannopoulou, and Dimitrios M. Thilikos. Forbidden graphs for tree-depth. *European Journal of Combinatorics*, 33(5):969–979, 2012. EuroComb '09. doi:10.1016/j.ejc.2011.09.014.
- 7 Martin Fürer and Huiwen Yu. Space saving by dynamic algebraization. In *9th International Computer Science Symposium in Russia, CSR 2014*, volume 8476 of *Lecture Notes in Computer Science*, pages 375–388. Springer, 2014. doi:10.1007/978-3-319-06686-8_29.
- 8 Jakub Gajarský, Stephan Kreutzer, Jaroslav Nešetřil, Patrice Ossona de Mendez, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. First-order interpretations of bounded expansion classes. *ACM Trans. Comput. Log.*, 21(4):29:1–29:41, 2020. doi:10.1145/3382093.
- 9 Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta theorems. In *AMS-ASL Joint Special Session on Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*, pages 181–206. American Mathematical Society, 2009.
- 10 Falko Hegerfeld and Stefan Kratsch. Solving connectivity problems parameterized by treedepth in single-exponential time and polynomial space. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020*, volume 154 of *LIPICs*, pages 29:1–29:16. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.STACS.2020.29.
- 11 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 184–192. IEEE, 2021. doi:10.1109/FOCS52979.2021.00026.
- 12 Łukasz Kowalik, Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge, and Piotr Wygocki. The PACE 2020 Parameterized Algorithms and Computational Experiments challenge: Treedepth. In *15th International Symposium on Parameterized and Exact Computation, IPEC 2020*, volume 180 of *LIPICs*, pages 37:1–37:18. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.IPEC.2020.37.
- 13 Wojciech Nadara, Michał Pilipczuk, and Marcin Smulewicz. Computing treedepth in polynomial space and linear fpt time. *CoRR*, abs/2205.02656, 2022. doi:10.48550/arXiv.2205.02656.
- 14 Jesper Nederlof, Michał Pilipczuk, Céline M. F. Swennenhuis, and Karol Węgrzycki. Hamiltonian Cycle parameterized by treedepth in single exponential time and polynomial space. In *46th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 12301 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2020. doi:10.1007/978-3-030-60440-0_3.
- 15 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.
- 16 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.
- 17 Jaroslav Nešetřil and Patrice Ossona de Mendez. On low tree-depth decompositions. *Graphs Comb.*, 31(6):1941–1963, 2015. doi:10.1007/s00373-015-1569-7.

- 18 Jaroslav Nešetřil and Patrice Ossona de Mendez. A distributed low tree-depth decomposition algorithm for bounded expansion classes. *Distributed Comput.*, 29(1):39–49, 2016. doi: 10.1007/s00446-015-0251-x.
- 19 Michael P. O’Brien and Blair D. Sullivan. Experimental evaluation of counting subgraph isomorphisms in classes of bounded expansion. *CoRR*, abs/1712.06690, 2017. arXiv:1712.06690.
- 20 Michał Pilipczuk and Sebastian Siebertz. Polynomial bounds for centered colorings on proper minor-closed graph classes. *J. Comb. Theory, Ser. B*, 151:111–147, 2021. doi: 10.1016/j.jctb.2021.06.002.
- 21 Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Parameterized circuit complexity of model-checking on sparse structures. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 789–798. ACM, 2018. doi: 10.1145/3209108.3209136.
- 22 Michał Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.
- 23 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In *41st International Colloquium on Automata, Languages, and Programming, ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014. doi:10.1007/978-3-662-43948-7_77.

The Pareto Cover Problem

Bento Natura  

London School of Economics and Political Science, UK

Meike Neuwöhner  

Forschungsinstitut für Diskrete Mathematik, Universität Bonn, Germany

Stefan Weltge  

Technische Universität München, Germany

Abstract

We introduce the problem of finding a set B of k points in $[0, 1]^n$ such that the expected cost of the cheapest point in B that dominates a random point from $[0, 1]^n$ is minimized. We study the case where the coordinates of the random points are independently distributed and the cost function is linear. This problem arises naturally in various application areas where customers' requests are satisfied based on predefined products, each corresponding to a subset of features. We show that the problem is NP-hard already for $k = 2$ when each coordinate is drawn from $\{0, 1\}$, and obtain an FPTAS for general fixed k under mild assumptions on the distributions.

2012 ACM Subject Classification Theory of computation \rightarrow Randomness, geometry and discrete structures; Theory of computation \rightarrow Dynamic programming; Theory of computation \rightarrow Packing and covering problems; Theory of computation \rightarrow Problems, reductions and completeness; Mathematics of computing \rightarrow Discrete optimization

Keywords and phrases Pareto, Covering, Optimization, Approximation Algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.80

Related Version *Full Version*: <https://arxiv.org/abs/2202.08035>

1 Introduction

Let $f: [0, 1]^n \rightarrow \mathbb{R}$ be a continuous cost function and $B \subseteq [0, 1]^n$ be a finite set. We consider the function

$$f_B: [0, 1]^n \rightarrow \mathbb{R} \cup \{\infty\}, \quad f_B(x) := \min\{f(b) : x \leq b, b \in B\},$$

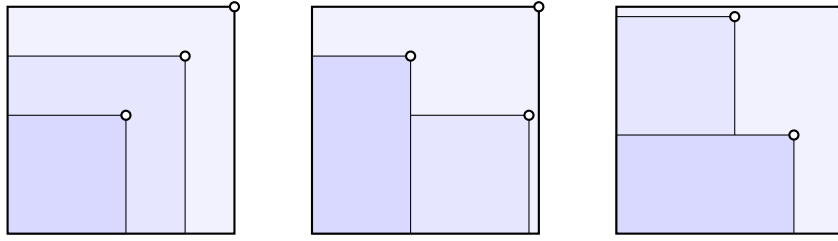
where we write $x \leq b$ if x is less or equal than b in every coordinate and say that b covers x . In other words, $f_B(x)$ is the smallest cost needed to cover x with a point from B . We say that B is a *Pareto cover*¹ of a probability measure μ on $[0, 1]^n$ if a random point can be covered by at least one point from B almost surely, i.e., $\mu(\{x \in [0, 1]^n : x \leq b \text{ for some } b \in B\}) = 1$. Note that B is always a Pareto cover if it contains the all-ones vector $\mathbf{1}$ (but for some probability measures, B is not required to contain $\mathbf{1}$).

Given f , μ , and an integer $k \geq 1$, we study the problem of finding a Pareto cover B of μ with $|B| = k$ such that $E_\mu[f_B]$ is minimized. That is, we are searching for a Pareto cover B of predefined size such that the expected cost of covering a random point with a point from B is smallest possible.

As an illustration, imagine a city with tourist attractions $[n] := \{1, \dots, n\}$ and suppose that the city wants to design k books B of vouchers for subsets of these attractions. Each tourist x will pick the cheapest book $b \in B$ that covers all attractions that x wants to visit. We can think of x as a binary vector in $\{0, 1\}^n$. Assuming that we have some probability

¹ In the context of multi-objective optimization, $x \leq b$ is commonly referred to as b *Pareto-dominates* x .





■ **Figure 1** For the Lebesgue measure (uniform distribution) on $[0, 1]^2$ with cost $f(x_1, x_2) = x_1 + x_2$, the optimal Pareto covers of size $k = 3$ are $\{(12/23, 12/23), (18/23, 18/23), (1, 1)\}$, $\{(10/23, 18/23), (22/23, 12/23), (1, 1)\}$, and $\{(18/23, 10/23), (12/23, 22/23), (1, 1)\}$.

distribution over the tourists x , we want to determine k books that, in expectation, cover the tourists requests in the cheapest way. Note that, in this example, μ is a discrete measure on $\{0, 1\}^n$. Defining an appropriate cost function, an optimal Pareto cover of size k is attained by a set of vectors in $\{0, 1\}^n$, each corresponding to a book.

Similar applications can be given for other areas where customer’s requests are satisfied based on predefined products, each corresponding to a subset of features. Note that our model also allows for non-binary requests $x \in [0, 1]^n \setminus \{0, 1\}^n$, which may correspond to features that are available in different quality ranges.

For another application, imagine a gang of robbers that wants to steal paintings in an art gallery. To this end, every gang member studies one painting $i \in [n]$ and estimates the probability p_i of being able to steal it. Their boss decides in advance which subset $S \subseteq [n]$ of paintings to steal. If all corresponding gang members are successful (assuming that they act independently), then the gang will receive a value $v(S)$. Otherwise, they all get caught and the gang receives $v(\emptyset) = 0$. The problem of finding a subset of paintings that maximizes the expected return can be modeled within the above framework as follows. Let μ be the probability measure on $\{0, 1\}^n$ that corresponds to setting each coordinate independently to 1 with probability $1 - p_i$, and set $f(x) := v([n]) - v(\{i \in [n] : x_i = 0\})$ for all $x \in \{0, 1\}^n$. Denoting by $B^* = \{b^*, \mathbf{1}\}$ an optimal Pareto cover of size $k = 2$, it is easy to see that $S = \{i \in [n] : b_i^* = 0\}$ maximizes the expected return.

Determining optimal Pareto covers of a given size is a difficult problem. Finding an analytical solution seems to be non-trivial even for very basic probability measures and cost functions, see Figure 1. In this work, we study the problem from the point of view of complexity theory. We particularly focus on product measures and linear cost functions, for which the problem is already hard as our first results show.

Here, for $A = \{a_0, \dots, a_{M+1}\}$ with $0 = a_0 < a_1 < \dots < a_M < a_{M+1} = 1$ and a vector $p = ((p_\ell^i)_{\ell=0}^{M+1})_{i=1}^n \in [0, 1]^{(M+2) \times n}$ with $\sum_{\ell=0}^{M+1} p_\ell^i = 1$ for all $i \in [n]$, let $\mu_{A,p}$ denote the discrete product measure on $[0, 1]^n$ where x_i assumes a_ℓ with probability p_ℓ^i , that is,

$$\mu_{A,p}(\{(a_{\ell_i})_{i=1}^n\}) = \prod_{i=1}^n p_{\ell_i}^i \text{ for all } (a_{\ell_i})_{i=1}^n \in A^n. \tag{1}$$

In addition, for $A = \{0, 1\}$ and $p \in [0, 1]^n$, we define the binary product measure $\mu_p := \mu_{\{0,1\},p}$ where $q_0^i = 1 - p_i$ and $q_1^i = p_i$ for $i \in [n]$, that is, x_i is set to $a_1 = 1$ with probability p_i , and to $a_0 = 0$ with probability $1 - p_i$. Finally, for a linear cost function f given by $f(x) = c^\top x$ for some $c \in \mathbb{R}^n$ and a finite set B , we overload our notation $c_B := f_B$.

The Pareto cover problem turns out to be computationally difficult even in a setting where we restrict ourselves to binary product measures and linear cost functions.

► **Theorem 1.** *Let $k \in \mathbb{Z}$, $k \geq 2$. Given $p \in ([0, 1] \cap \mathbb{Q})^n$, $c \in \mathbb{Q}^n$, $\gamma \in \mathbb{Q}$, the problem of deciding whether there is some Pareto cover B of μ_p such that $|B| = k$ and $E_{\mu_p}[c_B] \leq \gamma$ is weakly NP-complete for k constant. Moreover, there are values of $k \in \Theta(n)$ for which it is strongly NP-hard.*

If the size of the Pareto cover is part of the input, we do not know whether the corresponding problem is in NP. In fact, computing the objective value of a single Pareto cover is already difficult:

► **Proposition 2.** *Given a Pareto cover B for the uniform distribution μ on $\{0, 1\}^n$, the problem of computing $E_\mu[\mathbf{1}_B]$ is #P-hard.*

Even when k is constant, it is not immediate how to determine, in polynomial time, the objective value $E_{\mu_p}[c_B]$ a certain Pareto cover $B = \{b^1, \dots, b^k\}$ attains for given probabilities $p \in ([0, 1] \cap \mathbb{Q})^n$ and a positive cost vector $c \in \mathbb{Q}^n$. In this respect, observe that it is infeasible to simply sum over all vectors $x \in \{0, 1\}^n$ since there is an exponential number of them. However, for constant k , we can afford to iterate over all subsets of $[k]$ instead. Moreover, it is not hard to see that for every $J \subseteq [k]$, c_B is constant on the set X_J consisting of all vectors $x \in [0, 1]^n$ that are covered precisely by those vectors b^j for which $j \in J$.

We show how to employ dynamic programming in order to compute the values $\mu(X_J)_{J \subseteq [k]}$ for arbitrary discrete product measures μ on $[0, 1]^n$. Using this observation as a starting point, we manage to derive a fully polynomial-time approximation scheme (FPTAS) for the case where k is constant.

► **Theorem 3.** *Let $k \in \mathbb{N}$ be fixed. Given a discrete product measure μ on $[0, 1]^n$ and $c \in \mathbb{Q}_{\geq 0}^n$, the problem of computing an optimal Pareto cover of size k with respect to μ and c admits an FPTAS.*

We further show how to extend our approach to general product measures that satisfy some mild assumptions. Essentially, we will consider products of *nice* measures $(\mu_i)_{i=1}^n$ on $[0, 1]$ that allow us to efficiently query an approximation of $\mu_i((a, b])$ for each a, b, i as well as a positive lower bound on the expectation of the identity on $[0, 1]$ with respect to each μ_i . A more formal definition will be given later.

Our paper is structured as follows. In Section 2 we briefly discuss related work. The proofs of Theorem 1 and Proposition 2 are given in Section 3, where we also derive results for general discrete product measures that will be used in our FPTAS. The latter and hence the proof of Theorem 3 is presented in Section 4. The full version of this paper shows how to extend our FPTAS result to very general (product) measures. We close with some open questions in Section 5.

2 Related Work

To the best of our knowledge our setting for the general case $k > 2$ has not been studied in the literature. For the case $k = 2$ similar problems have been studied in the area of *stochastic optimization* in the context of *chance constrained optimization*. Here one aims to find an optimal solution to a problem with stochastic constraints. A solution to the problem then needs to fulfill the constraints with probability $1 - \delta$ for some $\delta > 0$.

Linear chance constrained problems are of the form

$$\min\{\langle c, x \rangle : x \in X, \mathbb{P}_{\xi \sim \mu}[Ax \geq \xi] \geq 1 - \delta\} \quad (2)$$

for a domain X , a distribution μ , matrix A and parameter δ . Note that our problem for $k = 2$ with linear cost functions f can be formulated as such a problem under the assumption that every Pareto cover has to contain $\mathbf{1}$. It remains to find the second vector in the optimal Pareto cover, for which one can guess the probability that it covers an element drawn from μ . This fits exactly into the framework of (2) where A is the identity matrix and $X = [0, 1]^n$. For an overview on the topic, we refer to the work of Nemirovski and Shapiro [4] and Luedtke, Ahmed, and Nemhauser [3]. In principle, it is possible to extend this idea to the case $k \geq 3$, for instance by using techniques from [3]. However, it is unclear whether theoretically efficient (approximation) algorithms can be obtained by such an approach.

The Pareto cover problem has an interesting interpretation in the context of tropical geometry. It can be equivalently phrased as a problem of partitioning $[0, 1]^n$ into k regions R_i . For each region, one selects the *tropical barycenter*, which is the coordinate-wise maximum of all its elements. This resembles a tropicalized version of classical Euclidean clustering algorithms and barycenters.

For randomized algorithms, tools from *statistical learning theory* (sample complexity, coresets, ...) can be used to study the Pareto cover problem. A natural approach would be to replace μ by a probability distribution $\tilde{\mu}$ that is a combination of polynomially many Dirac-distributions such that $\tilde{\mu}$ is “close” to μ , and reduce the problem to finding an optimal Pareto cover for $\tilde{\mu}$. In this work, however, we focus on deterministic algorithms and defer a discussion of such techniques to the full version of our paper.

3 Hardness results

In this section, we show that finding optimal Pareto covers (of given sizes) is a computationally hard problem, even for simple binary product measures and linear cost functions. In particular, we prove Theorem 1 and Proposition 2.

In the first part, we focus on the case where μ is a binary product measure given by some input vector $p \in [0, 1]^n$ such that $\mu = \mu_p$, see (1). We consider the following problem.

► **Definition 4.** The decision variant of the *binary Pareto cover problem* is defined as follows: Given $p \in ([0, 1] \cap \mathbb{Q})^n$, $c \in \mathbb{Q}_{\geq 0}^n$, $\gamma \in \mathbb{Q}$, and $k \in \mathbb{Z}_{\geq 1}$, decide whether there is some Pareto cover B of μ_p such that $|B| = k$ and $\mathbb{E}_{\mu_p}[c_B] \leq \gamma$.

We assume that p , c , and γ are given by their binary encodings. We leave open how k is encoded since in our applications it will be always polynomially bounded in n (see Proposition 2) or mostly be even a constant.

In Section 3.1, we show that the binary Pareto cover problem is weakly NP-hard if k is a fixed constant. In addition, we show that for $k = \frac{n+5}{3}$, the problem is strongly NP-hard. For the case that k is part of the input, we prove Proposition 2 showing that the problem of computing $\mathbb{E}_{\mu_p}[c_B]$ for a given Pareto cover B is #P-hard in Section 3.2. In view of this, it is unclear whether the binary Pareto cover problem is in NP if k is part of the input. However, for constant k , we establish in Section 3.3 that the problem is in NP, even for general discrete product measures. This result completes the proof of Theorem 1 and plays an important role in the design of our approximation algorithm.

3.1 NP-hardness

In this section, we consider the binary Pareto cover problem. Note that it can be solved efficiently in the case $k = 1$ since then $B = \{b\}$ is an optimal solution, where $b \in \{0, 1\}^n$ with $b_i = 1$ if and only if $p_i > 0$. In the remainder, we show that the binary Pareto cover

problem is weakly NP-hard for $k = 2$. Similar, but slightly different proofs for the cases $k = 3$ and $k \geq 4$ can be found in the full version of this paper. Moreover, our reduction for $k \geq 4$ proves strong NP-hardness for $k = \frac{n+5}{3}$.

In order to show that the binary Pareto cover problem is NP-hard for $k = 2$, let us recall the PARTITION problem [2], which is well-known to be (weakly) NP-hard: Given $a_1, \dots, a_n \in \mathbb{Z}_{\geq 1}$ with $\sum_{i=1}^n a_i$ even, decide whether there is a subset $I \subseteq [n]$ such that $\sum_{i \in I} a_i = \frac{1}{2} \sum_{i=1}^n a_i$. We provide a reduction from PARTITION to the binary Pareto cover problem with $k = 2$.

Before defining it precisely, let us describe the idea first. Given a PARTITION instance a_1, \dots, a_n , set $\alpha := \frac{2}{a_1 + \dots + a_n}$ and consider an instance of the binary Pareto cover problem with $p_i = 1 - e^{-\alpha a_i}$ and $c_i = a_i$ for all $i \in [n]$. Since all c_i and p_i are positive, every optimal Pareto cover of μ_p is of the form $B = \{b, \mathbf{1}\}$ where $b \in \{0, 1\}^n$. Setting $I := \{i \in [n] : b_i = 0\}$, we see that the cost $E_{\mu_p}[c_B]$ of B satisfies

$$\begin{aligned} \sum_{i=1}^n a_i - E_{\mu_p}[c_B] &= \mathbf{1}^\top c - E_{\mu_p}[c_B] \\ &= \mathbf{1}^\top c - c^\top b \cdot \mu_p(\{x \in \{0, 1\}^n : x \leq b\}) - c^\top \mathbf{1} \cdot \mu_p(\{x \in \{0, 1\}^n : x \not\leq b\}) \\ &= c^\top (\mathbf{1} - b) \cdot \mu_p(\{x \in \{0, 1\}^n : x \leq b\}) \\ &= \prod_{i \in I} (1 - p_i) \cdot \sum_{i \in I} a_i \\ &= e^{-\alpha \sum_{i \in I} a_i} \cdot \sum_{i \in I} a_i \\ &= h\left(\sum_{i \in I} a_i\right), \end{aligned}$$

where $h: \mathbb{R} \rightarrow \mathbb{R}$, $h(x) = x \cdot e^{-\alpha x}$. Since h has its unique maximum at $x = \alpha^{-1}$, we see that $B = \{b, \mathbf{1}\}$ is a Pareto cover with $E_{\mu_p}[c_B] \leq \sum_{i=1}^n a_i - h(\alpha^{-1})$ if and only if $I = \{i \in [n] : b_i = 0\}$ satisfies $\sum_{i \in I} a_i = \alpha^{-1} = \frac{1}{2} \sum_{i=1}^n a_i$. In other words, if a_1, \dots, a_n is a “yes” instance for PARTITION, then there is a Pareto B cover of size $k = 2$ of μ_p with cost $E_{\mu_p}[c_B] \leq \sum_{i=1}^n a_i - h(\frac{1}{2} \sum_{i=1}^n a_i)$. If a_1, \dots, a_n is a “no” instance, then every Pareto cover of size $k = 2$ will have cost $E_{\mu_p}[c_B] > \sum_{i=1}^n a_i - h(\frac{1}{2} \sum_{i=1}^n a_i)$.

Unfortunately, the probabilities that we have used in the above argument cannot be polynomially represented. However, we show that we can efficiently round them such that the above strategy still works. More specifically, our probabilities and the threshold cost γ will be defined as follows.

► **Lemma 5.** *Given $a_1, \dots, a_n \in \mathbb{Z}_{\geq 1}$, we can compute $p_1, \dots, p_n \in [0, 1]$ and $\gamma \in \mathbb{Q}$ in polynomial time such that*

$$\frac{1 - \beta}{e^{\alpha a_i}} \leq 1 - p_i \leq \frac{1 + \beta}{e^{\alpha a_i}} \quad \text{for all } i \in [n], \text{ and } \frac{(1 - \beta)^{n+2}}{\alpha \cdot e} \leq \sum_{i=1}^n a_i - \gamma \leq \frac{(1 - \beta)^n}{\alpha \cdot e},$$

where $\alpha := \frac{2}{\sum_{i=1}^n a_i}$ and $\beta := \frac{\alpha^2}{48(n+1)}$.

Let p_1, \dots, p_n and γ be given as in the above statement. Note that we have $0 < p_i \leq 1$ for all $i \in [n]$ since $p_i \leq 1 - \frac{1 - \beta}{e^{\alpha a_i}} \leq 1$ and

$$p_i \geq 1 - \frac{1 + \beta}{e^{\alpha a_i}} > 1 - \frac{1 + \beta}{1 + \alpha \cdot a_i} \geq 1 - \frac{1 + \beta}{1 + \alpha} > 0.$$

Finally, set $c_i := a_i$ for all $i \in [n]$. It remains to prove the following lemma.

► **Lemma 6.** *There is a subset $I \subseteq [n]$ with $\sum_{i \in I} a_i = \frac{1}{2} \sum_{i=1}^n a_i$ if and only if there is a Pareto cover B of μ_p with $|B| = 2$ and $E_{\mu_p}[c_B] \leq \gamma$.*

The proves of the above lemmas can be found in the full version.

Note that we have shown that the binary Pareto cover problem is weakly NP-hard for constant k . We remark that, unless $P = NP$, the problem cannot be strongly NP-hard, as we derive an FPTAS in Section 4.

3.2 #P-hardness

In the previous section we have seen that the binary Pareto cover problem is NP-hard, already for constant k . The next natural question is whether the problem is in NP. At first sight, a Pareto cover $B \subseteq \{0, 1\}^n$ itself seems to be a canonical certificate for a “yes” instance. However, with this choice, we should be able to compute the cost $E_{\mu_p}[c_B]$ efficiently. Unfortunately, if k is not part of the input, i.e., B is not of constant size, computing $E_{\mu_p}[c_B]$ is hard. More precisely, let us prove Proposition 2, which states that, given a Pareto cover B for the uniform distribution μ on $\{0, 1\}^n$, the problem of computing $E_{\mu}[\mathbf{1}_B]$ is #P-hard.

Proof of Proposition 2. We use the fact that the problem of computing the number of vertex covers in a given undirected graph is #P-hard [5]. Given a graph $G = (V, E)$, identify V with $[n]$ and for every edge $e \in E$, let b_e denote the characteristic vector of $V \setminus e$, the set of nodes that are not part of e . Let μ denote the uniform distribution on $\{0, 1\}^n$, i.e., $\mu = \mu_p$ with $p = \frac{1}{2} \cdot \mathbf{1}$. Consider the Pareto cover $B := \{b_e : e \in E\} \cup \{\mathbf{1}\}$. Setting $c = \mathbf{1}$, the cost of B is equal to

$$\begin{aligned} E_{\mu}[c_B] &= (n-2) \cdot \mu(\{x \in \{0, 1\}^n : x \leq b_e \text{ for some } e \in E\}) \\ &\quad + n \cdot \mu(\{x \in \{0, 1\}^n : x \not\leq b_e \text{ for all } e \in E\}) \\ &= (n-2) + 2 \cdot \mu(\{x \in \{0, 1\}^n : x \not\leq b_e \text{ for all } e \in E\}) \\ &= (n-2) + \frac{|\{x \in \{0, 1\}^n : x \not\leq b_e \text{ for all } e \in E\}|}{2^{n-1}} \\ &= (n-2) + \frac{|\{U \subseteq [n] : U \cap e \neq \emptyset \text{ for all } e \in E\}|}{2^{n-1}}, \end{aligned}$$

and hence we see that $2^{n-1} \cdot (E_{\mu}[c_B] - (n-2))$ is the number of vertex covers in G . ◀

3.3 Membership in NP for constant k

For the binary Pareto cover problem, we have seen that computing the cost of a given Pareto cover B is hard if B can be of any size. In this section, we show that the cost can be computed efficiently if B is of constant size. In fact, we prove that this is the case for the discrete version of our problem (see (1)).

We introduce the following notation: For probability measures $(\mu_i)_{i=1}^n$ defined on (the Borel σ -algebra on) $[0, 1]$, we define their product $\mu := \prod_{i=1}^n \mu_i$ to be given by $\mu(I_1 \times \dots \times I_n) = \prod_{i=1}^n \mu_i(I_i)$, where $(I_i)_{i=1}^n$ are intervals contained in $[0, 1]$. In particular, for $A = \{a_0, \dots, a_{M+1}\}$ and $p = ((p_j^i)_{j=0}^{M+1})_{i=1}^n$ as in (1), $\mu = \mu_{A,p}$ is the product of the measures $\mu_i = \mu_{i,A,p}$ given by $\mu_i(\{a_j\}) = p_j^i$, $i \in [n]$, $j = 0, \dots, M+1$. We study the following problem.

► **Definition 7.** The decision variant of the *discrete Pareto cover problem* is the following: Given $A = \{a_0, \dots, a_{M+1}\}$ with $0 = a_0 < a_1 < \dots < a_M < a_{M+1} = 1$, $p = ((p_j^i)_{j=0}^{M+1})_{i=1}^n$ with $p_j^i \in [0, 1]$ and $\sum_{j=0}^{M+1} p_j^i = 1$ for all i , $c \in \mathbb{Q}_{\geq 0}^n$, $\gamma \in \mathbb{Q}$, and $k \in \mathbb{Z}_{\geq 1}$, decide whether there is some Pareto cover B of $\mu_{A,p}$ such that $|B| = k$ and $E_{\mu_p}[c_B] \leq \gamma$.

Again, we assume that A , p , c , and γ are given by their binary encodings. In our applications, k will be always constant.

Note that it is easy to check whether a finite set $B \subseteq [0, 1]^n$ is feasible for the above problem, i.e., that it is a Pareto cover of $\mu_{A,p}$. In fact, let $x^* \in [0, 1]^n$ be given by $x_i^* := \max\{a_j : p_j^i > 0\}$ for $i \in [n]$. Then B is a Pareto cover of $\mu_{A,p}$ if and only if $|B| = k$ and B contains at least one point that covers x^* .

Moreover, note that if B is feasible for the above problem, then we may assume that $B \subseteq A^n$ holds since otherwise we may lower entries of points in B without changing the set of points they cover and without increasing their cost.

► **Proposition 8.** *Let $k \in \mathbb{Z}_{\geq 1}$ be fixed. Given A, p, c as in the discrete Pareto cover problem and a Pareto cover $B \subseteq A^n$ for $\mu = \mu_{A,p}$ with $|B| = k$, we can compute $E_\mu[c_B]$ in polynomial time.*

Note that this shows that the discrete Pareto cover problem (and hence also the binary Pareto cover problem) is in NP if k is constant.

In order to prove Proposition 8, we make use of the following notation. For $i \in [n]$ vectors $b^1, \dots, b^k, x \in [0, 1]^n$, we define $J^i(x)$ to consist of all indices $j \in [k]$ such that b^j covers x , if we restrict both vectors to the first i coordinates. More precisely, we set

$$J^i(x) := \left\{ j \in [k] : x_1 \leq b_1^j, \dots, x_i \leq b_i^j \right\} \text{ for } i \in [n], \text{ and let } J(x) := J^n(x).$$

Whenever we refer to $J(x), J^1(x), \dots, J^n(x)$, the vectors b^1, \dots, b^k will be clear from the context. Observe that for $i \in [n]$ and $J \subseteq [k]$, the set $\{x \in [0, 1]^n : J^i(x) = J\}$ is a Borel set.

Note that $\{b^1, \dots, b^k\}$ is a Pareto cover for μ if and only if $\mu(\{x \in [0, 1]^n : J(x) = \emptyset\}) = 0$. Let us rephrase the cost of a Pareto cover using this new notation:

► **Lemma 9.** *Let μ be a probability measure on (the Borel σ -algebra on) $[0, 1]^n$ and let $B = \{b^1, \dots, b^k\}$ be a Pareto cover of μ . Then for every $c \in \mathbb{R}^n$ we have*

$$E_\mu[c_B] = \sum_{\emptyset \neq J \subseteq [k]} \mu(\{x \in [0, 1]^n : J(x) = J\}) \cdot \min_{j \in J} c^\top b^j.$$

Proof. For $J \subseteq [k]$, note that $c_B(x) = \min_{j \in J} c^\top b^j$ holds for all $x \in [0, 1]^n$ with $J(x) = J$. The claim follows since the sets $(\{x \in [0, 1]^n : J(x) = J\})_{J \subseteq [k]}$ are disjoint. ◀

Thus, in order to prove Proposition 8, it suffices to show that we can compute the values $\mu_{A,p}(\{x \in [0, 1]^n : J(x) = J\})$ for all $J \subseteq [k]$ in polynomial time. To this end, we show how to iteratively compute the values $\mu_{A,p}(\{x \in [0, 1]^n : J^i(x) = J\})$ for all $J \subseteq [k]$ and $i \in [n]$. Lemma 10 takes care of the base case $i = 1$, whereas Lemma 11 explains how to proceed from i to $i + 1$.

► **Lemma 10.** *Let μ_1, \dots, μ_n be probability measures on $[0, 1]$ and let $b^1, \dots, b^k \in [0, 1]^n$. For $\mu = \prod_{i=1}^n \mu_i$ and $J \subseteq [k]$ we have*

$$\mu(\{x \in [0, 1]^n : J^1(x) = J\}) = \mu_1((\alpha, \beta] \cap [0, 1]),$$

where $\alpha = \max_{j \in [k] \setminus J} b_1^j$ and $\beta = \min_{j \in J} b_1^j$.

Here, we use the convention $\max \emptyset := -\infty$ and $\min \emptyset := +\infty$.

Proof of Lemma 10. We have $J^1(x) = J$ if and only if $x_1 \leq b_1^j$ for all $j \in J$ and $x_1 > b_1^j$ for all $j \in [k] \setminus J$. That is, $\{x \in [0, 1]^n : J^1(x) = J\} = \{x \in [0, 1]^n : x_1 \in (\alpha, \beta]\}$. ◀

► **Lemma 11.** Let μ_1, \dots, μ_n be probability measures on $[0, 1]$ and let $b^1, \dots, b^k \in [0, 1]^n$. For $\mu = \prod_{i=1}^n \mu_i$, $J \subseteq [k]$, and $i \in \{2, \dots, n\}$ we have

$$\mu(\{x \in [0, 1]^n : J^i(x) = J\}) = \sum_{J \subseteq L \subseteq [k]} \mu(\{x \in [0, 1]^n : J^{i-1}(x) = L\}) \cdot \mu_i((\alpha_L, \beta_L] \cap [0, 1]),$$

where $\alpha_L = \max_{j \in L \setminus J} b_i^j$ and $\beta_L = \min_{j \in J} b_i^j$.

Proof. The claim follows from the fact that $\{x \in [0, 1]^n : J^i(x) = J\}$ is equal to

$$\bigcup_{J \subseteq L \subseteq [k]} \left[\{x \in [0, 1]^n : J^{i-1}(x) = L\} \cap \{x \in [0, 1]^n : x_i \leq b_i^j \text{ for all } j \in J, x_i > b_i^j \text{ for all } j \in L \setminus J\} \right]$$

and the observation that the above sets are disjoint. ◀

Note that for measures $\mu = \mu_{A,p} = \prod_{i=1}^n \mu_i$ and $\alpha, \beta \in \mathbb{Q} \cup \{\pm\infty\}$, we can compute $\mu_i((\alpha, \beta] \cap [0, 1])$ in polynomial time. Moreover, for constant k , the sum in Lemma 11 only has a constant number of summands. This yields Proposition 8.

4 Approximation algorithm

The goal of this section is to develop an FPTAS (see [1]) for the discrete Pareto cover problem, see Theorem 3. More precisely, we provide an algorithm that receives an instance I of the discrete Pareto cover problem and a parameter $\gamma \in (0, 1) \cap \mathbb{Q}$, and computes a $(1 + \gamma)$ -approximate solution to I in time polynomial in γ^{-1} and the encoding length of I . All proofs and an extension of our FPTAS to more general product measures can be found in the full version.

Let A, p, c, k define an instance of the discrete Pareto cover problem, where k is a constant. For every $i \in [n]$ let $l_i := \max\{l : p_l^i > 0\}$ and define $a^* := (a_{l_1}, \dots, a_{l_n})$. Recall that every Pareto cover B of $\mu = \mu_{A,p}$ must contain a point that covers a^* . Conversely, every finite set $B \subseteq [0, 1]^n$ containing a^* is a Pareto cover of μ . Since the costs are non-negative, we can restrict ourselves to Pareto covers that contain a^* .

Next, we discuss how to determine a cover of approximately minimum cost. Recall that in Section 3.3, Lemma 9, given a Pareto cover $B = \{b^1, \dots, b^k\}$, we have seen that we can express our objective as

$$E_\mu[c_B] = \sum_{\emptyset \neq J \subseteq [k]} \mu(\{x \in [0, 1]^n : J(x) = J\}) \cdot \min_{j \in J} c(b^j).$$

Even more, we know that we can iteratively compute the values $\mu(\{x \in [0, 1]^n : J^i(x) = J\})$ for $i \in [n]$ and $J \subseteq [k]$ in polynomial time, see Lemma 11. In doing so, the only information we need to proceed from i to $i + 1$ are the probabilities $\mu(\{x \in [0, 1]^n : J^i(x) = J\})$ for $J \subseteq [k]$ and the values $(b_{i+1}^j)_{j=1}^k$, but *no* further information on the values b_l^j for $l \in [i]$ and $j \in [k]$. What is more, by definition, the values $\mu(\{x \in [0, 1]^n : J^i(x) = J\})$ for $J \subseteq [k]$ and $\sum_{l=1}^i c_l \cdot b_l^j$ for $j \in [k]$ do not depend on the coordinates b_l^j , $l = i + 1, \dots, k$, $j \in [k]$. All in all, these are the best preconditions for a dynamic programming approach and motivate the following definition:

► **Definition 12.** For a Pareto cover $B = (b^j)_{j=1}^k$ with $b^k = a^*$ and $i \in [n]$ let

$$\text{Cand}^i(B) := (i, (P_J)_{J \subseteq [k]}, (C_j)_{j=1}^k),$$

where $P_J = \mu(\{x \in [0, 1]^n : J^i(x) = J\})$ for $J \subseteq [k]$ and $C_j = \sum_{l=1}^i c_l \cdot b_l^j$ for $j \in [k]$.

► **Definition 13.** A *candidate* is a triple $\mathcal{C} = (i, (P_J)_{J \subseteq [k]}, (C_j)_{j=1}^k)$.

The *cost* of the candidate \mathcal{C} is given by $\text{Cost}(\mathcal{C}) := \sum_{\emptyset \neq J \subseteq [k]} P_J \cdot \min_{j \in J} C_j$.

We call \mathcal{C} *valid* if there exists a Pareto cover $B = (b^j)_{j=1}^k$ with $b^k = a^*$ such that $\mathcal{C} = \text{Cand}^i(B)$, and say that B *witnesses* the validity of the candidate.

Note that the definition of the cost of a candidate is in accordance with Lemma 9.

A naive approach to tackle the discrete Pareto cover problem would now be to iteratively enumerate all valid candidates for $i = 1, \dots, n$, select a candidate for $i = n$ that yields the minimum objective value, and then back-trace to compute a corresponding cover. The problem with this idea is of course that we do not have a polynomial bound on the number of candidates we generate. To overcome this issue, we round the candidates appropriately to ensure a polynomial number of possible configurations, whilst staying close enough to the original values to obtain a good approximation of the objective for $i = n$. Observe that for constant k , the number of entries of each candidate is constant, which means that it suffices to polynomially bound the number of values each of them may attain.

To this end, consider Algorithm 1. The gray lines are not part of the algorithm itself, but only needed for its analysis. Before diving into the analysis of Algorithm 1, we would like to provide some intuition about what is happening. We start by enumerating all possible values $(b_1^j)_{j=1}^k$ may attain in a solution B with $b^k = a^*$ and use this information to compute $\text{Cand}^1(B)$ according to Definition 12. (Recall that this is independent of the values b_l^j for $l \geq 2, j = 1, \dots, k$.) Then, we round all non-zero entries of $\text{Cand}^1(B)$ (except for the first one, which is 1) down to the next power of $1 + \delta$, $\delta = \frac{\epsilon}{4n}$. Each rounded candidate \mathcal{C} is added to our table \mathcal{T} , and for back-tracing purposes, we store a cover B that leads to \mathcal{C} as $\text{Witness}(\mathcal{C})$. For the analysis, we further maintain an imaginary map AllWits mapping each rounded candidate $\mathcal{C} \in \mathcal{T}$ to the set of all possible witness covers that result in \mathcal{C} after (iterative) rounding.

After dealing with the base case $i = 1$, we enumerate possible values of $(b_i^j)_{j=1}^k$ for $i = 2, \dots, n$, loop over all rounded candidates for $i - 1$ and compute new rounded candidates for i according to Lemma 11. Moreover, we deduce witnesses for our new candidates for i from those stored for the candidates for $i - 1$ and the values $(b_i^j)_{j=1}^k$. This might of course lead to an exponential growth of the size of the imaginary map AllWits . However, the fact that the Witness-map only memorizes one witness per candidate keeps the total running time under control, provided we can come up with a polynomial bound on the number of candidates we generate. Lemma 14 takes care of this, and is the main ingredient of the proof of Theorem 15, which guarantees a polynomial running time.

► **Lemma 14.** *At each point during the algorithm, we have $|\mathcal{T}| \leq \alpha^{2^k} \cdot \beta^k \cdot n$, where*

$$\alpha = n + 2 - n \cdot \min \{ \log_{1+\delta}(p_l^i) : i \in [n], l \in \{0, \dots, M + 1\}, p_l^i > 0 \},$$

$$\beta = n + 2 + \log_{1+\delta}(c_1 + \dots + c_n) - \log_{1+\delta}(a_1) - \min \{ \log_{1+\delta}(c_i) : i \in [n] \}.$$

In particular, for constant k , $|\mathcal{T}|$ is polynomially bounded in the encoding length of the given instance of the discrete Pareto cover problem and ϵ^{-1} .

► **Theorem 15.** *Given an instance I of the discrete Pareto cover problem and a parameter $\epsilon \in (0, 1) \cap \mathbb{Q}$ as input, Algorithm 1 runs in time polynomial in $\text{size}(I)$ and ϵ^{-1} .*

Denote the set of all candidate in \mathcal{T} starting with i by \mathcal{T}_i . In order to finally obtain an FPTAS for the discrete Pareto cover problem, our goal for the remainder of this section is to prove the following result:

► **Theorem 16.** *Running Algorithm 1 and choosing the witness of a candidate of minimum cost in \mathcal{T}_n yields a $(1 + \epsilon)$ -approximation.*

■ **Algorithm 1** Dynamic program to compute rounded candidates.

Input: $(a_l)_{l=0}^{M+1}$, $((p_l^i)_{l=0}^{M+1})_{i=1}^n$, $(c_i)_{i=1}^n$, k , $\epsilon \in \mathbb{Q} \cap (0, 1)$
Output: a table \mathcal{T} of rounded candidates

- 1
- 2 For $i = 1, \dots, n$ compute $l_i := \max\{l : p_l^i > 0\}$.
- 3 $a^* \leftarrow (a_{l_i})_{i=1}^n$, $A \leftarrow \{a_0, \dots, a_{M+1}\}$, $\mathcal{T} \leftarrow \emptyset$
- 4 $\text{AllWits}(-) \leftarrow \emptyset$
- 5 $\delta \leftarrow \frac{\epsilon}{4n}$
- 6 **foreach** $(\beta_j)_{j=1}^k \in A^k$ with $\beta_k = a_1^*$ **do**
- 7 Define $(P_J)_{J \subseteq [k]}$ by $P_J \leftarrow \mu_1((\max_{j \in [k] \setminus J} \beta_j, \min_{j \in J} \beta_j) \cap [0, 1])$
- 8 $P_J \leftarrow \begin{cases} (1 + \delta)^{\lfloor \log_{1+\delta} P_J \rfloor} & , P_J > 0 \\ 0 & , P_J = 0 \end{cases}$
- 9 Define $(C_j)_{j=1}^k$ by $C_j \leftarrow \begin{cases} (1 + \delta)^{\lfloor \log_{1+\delta}(c_1 \cdot \beta_j) \rfloor} & , c_1 \cdot \beta_j > 0 \\ 0 & , c_1 \cdot \beta_j = 0 \end{cases}$
- 10 $\mathcal{T} \leftarrow \mathcal{T} \cup \{(1, (P_J)_{J \subseteq [k]}, (C_j)_{j=1}^k)\}$
- 11 $b_1^j \leftarrow \beta_j$, $j = 1, \dots, k$, $b_i^j \leftarrow 0$, $i = 2, \dots, n$, $j = 1, \dots, k - 1$, $b_i^k \leftarrow a_i^*$, $i = 2, \dots, n$
- 12 $\text{Witness}((1, (P_J)_{J \subseteq [k]}, (C_j)_{j=1}^k)) \leftarrow (b^j)_{j=1}^k$
- 13 $\text{AllWits}((1, (P_J)_{J \subseteq [k]}, (C_j)_{j=1}^k)) \leftarrow \text{AllWits}((1, (P_J)_{J \subseteq [k]}, (C_j)_{j=1}^k)) \cup \{(b^j)_{j=1}^k\}$
- 14 **for** $i = 2$ **to** n **do**
- 15 **foreach** $(i - 1, (P_J^{i-1})_{J \subseteq [k]}, (C_j^{i-1})_{j=1}^k) \in \mathcal{T}$ **do**
- 16 **foreach** $(\beta_j)_{j=1}^k \in A^k$ with $\beta_k = a_i^*$ **do**
- 17 Define $(P_J^i)_{J \subseteq [k]}$ by
- 18 $P_J^i \leftarrow \sum_{J \subseteq L \subseteq [k]} P_L^{i-1} \cdot \mu_i((\max_{j \in L \setminus J} \beta_j, \min_{j \in J} \beta_j) \cap [0, 1])$
- 19 $P_J^i \leftarrow \begin{cases} (1 + \delta)^{\lfloor \log_{1+\delta} P_J^i \rfloor} & , P_J^i > 0 \\ 0 & , P_J^i = 0 \end{cases}$
- 20 Define $(C_j^i)_{j=1}^k$ by $C_j^i \leftarrow C_j^{i-1} + c_i \cdot \beta_j$
- 21 $C_j^i \leftarrow \begin{cases} (1 + \delta)^{\lfloor \log_{1+\delta} C_j^i \rfloor} & , C_j^i > 0 \\ 0 & , C_j^i = 0 \end{cases}$
- 22 $\mathcal{T} \leftarrow \mathcal{T} \cup \{(i, (P_J^i)_{J \subseteq [k]}, (C_j^i)_{j=1}^k)\}$
- 23 $(b^{i-1, j})_{j=1}^k \leftarrow \text{Witness}((i - 1, (P_J^{i-1})_{J \subseteq [k]}, (C_j^{i-1})_{j=1}^k))$
- 24 Define $(b^{i, j})_{j=1}^k$ by $b_l^{i, j} := \begin{cases} b_l^{i-1, j} & , l \neq i \\ \beta_j & , l = i \end{cases}$
- 25 $\text{Witness}((i, (P_J^i)_{J \subseteq [k]}, (C_j^i)_{j=1}^k)) \leftarrow (b^{i, j})_{j=1}^k$
- 26 **foreach** $(\tilde{b}^{i-1, j})_{j=1}^k \in \text{AllWits}((i - 1, (P_J^{i-1})_{J \subseteq [k]}, (C_j^{i-1})_{j=1}^k))$ **do**
- 27 Define $(\tilde{b}^{i, j})_{j=1}^k$ by $\tilde{b}_l^{i, j} := \begin{cases} \tilde{b}_l^{i-1, j} & , l \neq i \\ \beta_j & , l = i \end{cases}$
- 28 $\text{AllWits}((i, (P_J^i)_{J \subseteq [k]}, (C_j^i)_{j=1}^k)) \leftarrow$
 $\text{AllWits}((i, (P_J^i)_{J \subseteq [k]}, (C_j^i)_{j=1}^k)) \cup \{(\tilde{b}^{i, j})_{j=1}^k\}$

28 **return** \mathcal{T}

The proof of Theorem 16 consists of two main steps. Lemma 17 shows that any rounded candidate \tilde{C} we store in \mathcal{T} invokes similar costs to those of any of its witness covers. Proposition 18 ensures that every cover $B = (b^j)_{j=1}^k$ with $b^k = a^*$ occurs as a possible witness for some candidate. In particular, this holds for an optimum cover B^* (with $b^{*,k} = a^*$) and by Lemma 17, we can therefore infer that the costs of the solution we return can only be by a factor of $1 + \epsilon$ larger than the optimum.

► **Lemma 17.** *Let $\tilde{C} \in \mathcal{T}_n$ and let $B \in \text{AllWits}(\tilde{C})$. Then $\text{Cost}(\tilde{C}) \leq E_\mu[c_B] \leq (1 + \epsilon) \cdot \text{Cost}(\tilde{C})$.*

► **Proposition 18.** *Let an instance of the discrete Pareto cover problem be given. For each $i \in [n]$ and each cover $B = (b^j)_{j=1}^k$ such that $b^k = a^*$ and $b_l^j = 0$ for $j = 1 \dots, k - 1$ and $l = i + 1, \dots, n$, there exists $\mathcal{C} \in \mathcal{T}_i$ such that $B \in \text{AllWits}(\mathcal{C})$.*

Combining Theorem 16, Lemma 14 and Theorem 15 and observing that we can compute the cost of a candidate in polynomial time for constant k , we obtain Theorem 3, which we restate once again:

► **Theorem 3.** *Let $k \in \mathbb{N}$ be fixed. Given a discrete product measure μ on $[0, 1]^n$ and $c \in \mathbb{Q}_{\geq 0}^n$, the problem of computing an optimal Pareto cover of size k with respect to μ and c admits an FPTAS.*

5 Conclusion

In this paper, we have introduced the Pareto cover problem and studied the case of product measures and linear cost functions. For fixed k , we have come to a pretty good understanding of its complexity: On the one hand, we could show weak NP-hardness of the problem. On the other hand, we have established the existence of an FPTAS.

However, there are several questions that remain open and constitute an interesting subject for future research. To begin with, we have seen that even in a very restricted setting such as the uniform probability distribution on $[0, 1]^n$, it seems non-trivial to find an optimum cover (see Figure 1). Consequently, in order to obtain a better feeling for the problem at hand, it can be worthwhile to examine the structure of optimum solutions for such special cases.

When dealing with the binary problem variant, another question that comes up is for which subsets of $\{0, 1\}^n$, there exists an instance they are optimum for. Any insights towards this question may lead to new, perhaps more efficient strategies to tackle the Pareto cover problem.

In addition to these rather concrete questions, there are also several more fundamental issues one may want to address. For instance, even though we have seen that computing the objective value attained by an arbitrary solution is #P-hard in general (i.e., for non-fixed k), this does not resolve containment in NP since there might be another choice of a certificate that does the trick. More generally, it would be interesting to fully understand the dependence of the problem complexity on the parameter k . To this end, note that the complexity does not simply “increase” with larger values of k , given that in the discrete setting, the problem is weakly NP-hard for constant $k \geq 2$, and strongly NP-hard, e.g., for $k = \frac{n+5}{3}$, but once k is at least as large as the number of all possible discrete vectors b , it is obvious what an optimum solution should look like (and we can output it in polynomial time, assuming an appropriate output encoding is chosen). Hence, it seems interesting to further investigate the hardness transition of the problem: When exactly does the problem become strongly NP-hard? When does it become easier again?

Finally, as all of our results apply to the case of product measures, it appears natural to ask what can be done for general probability measures. To this end, as alluded to in Section 2, it could be fruitful to explore connections to existing results from statistical learning theory.

References

- 1 Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer Science & Business Media, 1999. doi:10.1007/978-3-642-58412-1.
- 2 Michael R Garey and David S Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990. URL: <https://dl.acm.org/doi/10.5555/574848>.
- 3 James Luedtke, Shabbir Ahmed, and George L Nemhauser. An integer programming approach for linear programs with probabilistic constraints. *Mathematical programming*, 122(2):247–272, 2010. doi:10.1007/s10107-008-0247-4.
- 4 Arkadi Nemirovski and Alexander Shapiro. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17(4):969–996, 2007. doi:10.1137/050622328.
- 5 Salil P Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001. doi:10.1137/S0097539797321602.

A Unified Framework for Hopsets

Ofer Neiman 

Ben-Gurion University of the Negev, Beer-Sheva, Israel

Idan Shabat 

Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

Given an undirected graph $G = (V, E)$, an (α, β) -hopset is a graph $H = (V, E')$, so that adding its edges to G guarantees every pair has an α -approximate shortest path that has at most β edges (hops), that is, $d_G(u, v) \leq d_{G \cup H}^{(\beta)}(u, v) \leq \alpha \cdot d_G(u, v)$. Given the usefulness of hopsets for fundamental algorithmic tasks, several different algorithms and techniques were developed for their construction, for various regimes of the stretch parameter α .

In this work we devise a single algorithm that can attain all state-of-the-art hopsets for general graphs, by choosing the appropriate input parameters. In fact, in some cases it also improves upon the previous best results. We also show a lower bound on our algorithm.

In [3], given a parameter k , a $(O(k^\epsilon), O(k^{1-\epsilon}))$ -hopset of size $\tilde{O}(n^{1+1/k})$ was shown for any n -vertex graph and parameter $0 < \epsilon < 1$, and they asked whether this result is best possible. We resolve this open problem, showing that any (α, β) -hopset of size $O(n^{1+1/k})$ must have $\alpha \cdot \beta \geq \Omega(k)$.

2012 ACM Subject Classification Theory of computation \rightarrow Shortest paths

Keywords and phrases Graph Algorithms, Shortest Paths, Hopsets

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.81

Related Version *Full Version:* <https://arxiv.org/pdf/2108.09673.pdf>

Funding Partially supported by the Lynn and William Frankel Center for Computer Sciences and ISF grant 970/21.

Acknowledgements We want to acknowledge the review of the anonymous reviewers from SODA 2022. This review drew our attention for some improvements that we applied in the paper.

1 Introduction

Hopsets are graph theoretic structures that have gained much attention recently [5, 20, 14, 13, 8, 1, 10, 15, 7, 3]. They play a role in central algorithmic applications such as approximating shortest paths [16, 5, 2, 11], distributed computing tasks [9, 18, 4, 6], dynamic graph algorithms [14, 17], and many more.

Given a graph $G = (V, E)$, possibly with non-negative weights on the edges $w : E \rightarrow \mathbb{R}$, an (α, β) -hopset is a graph $H = (V, E')$ such that every pair in V has an α -approximate shortest path in $G \cup H$ with at most β hops. That is, for all $u, v \in V$,

$$d_G(u, v) \leq d_{G \cup H}^{(\beta)}(u, v) \leq \alpha \cdot d_G(u, v),$$

where $d_G(u, v)$ is the distance between u, v in G , and $d_{G \cup H}^{(\beta)}(u, v)$ stands for the length of the shortest path in $G \cup H$ between u, v that has at most β edges. The weight of an edge $(x, y) \in E'$ of H is defined to be the length of the shortest path in G that connects x and y .

Hopsets were first introduced by [5], although they were implicitly used before in [16]. In her seminal work, given a parameter k that determines the hopset size, [5] devised a construction of $(1 + \epsilon, \beta)$ -hopsets of size $O(n^{1+1/k} \cdot \log n)$ with $\beta = O\left(\frac{\log n}{\epsilon}\right)^{\log k}$. This result was recently improved by [8, 15, 10], who obtained $\beta = O\left(\frac{\log k}{\epsilon}\right)^{\log k}$ and size $O(n^{1+1/k})$.



© Ofer Neiman and Idan Shabat;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 81; pp. 81:1–81:13



Leibniz International Proceedings in Informatics

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

On the opposite end of the stretch spectrum, for a stretch factor linear in k , it is folklore that the distance oracle of [21] (henceforth the TZ algorithm) is in fact a $(2k - 1, 2)$ -hopset of size $O(k \cdot n^{1+1/k})$.

A lower bound of [1] asserts that any $(1 + \epsilon, \beta)$ -hopset of size $O(n^{1+1/k})$ must have $\beta = \Omega\left(\frac{1}{\epsilon \log k}\right)^{\log k}$. This lower bound is meaningful only when the stretch is smaller than $1 + 1/\log k$, so it motivates the natural question: allowing the stretch to be larger than $1 + 1/\log k$, what is the trade off between stretch and hopbound?

This question was partially studied by [7, 3], who showed $(3 + \epsilon, \beta)$ -hopsets of size $O(n^{1+1/k} \cdot \log \Lambda)$ with improved $\beta = k^{\log(3+O(1/\epsilon))}$, where Λ is the aspect ratio of the graph¹ (In fact, [7] did not have the $\log \Lambda$ factor in the size, albeit their β had a somewhat worse exponent). More generally, for any $0 < \epsilon < 1$, [3] devised a $(O(k^\epsilon), O_\epsilon(k^{1-\epsilon}))$ -hopset of size $O(k^\epsilon \cdot n^{1+1/k} \cdot \log \Lambda)$. We note that by choosing $\epsilon = O(\frac{1}{\log k})$ they get a $(O(c), k^{1+O(1/\log c)})$ -hopset for any constant $c > 1$. The previous state-of-the-art results for hopsets are summarized in Table 1.

There are two main concerns with the current state of affairs regarding hopsets. First, there is no lower bound for any constant (or larger) stretch. Indeed, the tightness of the $(O(k^\epsilon), O_\epsilon(k^{1-\epsilon}))$ -hopset was asked as an open question in [3]. The second concern is that previous hopset constructions use a variety of different techniques for each possible range of the stretch α : from the sparse covers used by [5], to two types of the TZ sampling algorithm [21, 22], the superclustering technique of [12], and in some cases a certain combination of these with other ingredients. For instance, the algorithms of [3] for hopsets with stretch $3 + \epsilon$ and $O(k^\epsilon)$ are rather complicated, and contain a three-stage construction, involving a truncated application of the [22] algorithm, a superclustering phase, and a multiplicative spanner built on some cluster graph.

In this paper we devise a single framework that unifies all previous results for hopsets, matching and even improving upon the state-of-the-art in all the possible stretch regimes. In addition, we answer affirmatively the question of [3] mentioned above.

■ **Table 1** Previous results on (α, β) -hopsets for n -vertex weighted graphs, with parameter $k \geq 1$ (the dependence on k in the size is omitted for brevity).

Stretch	Hopbound	Hopset Size	Paper
$1 + \epsilon$	$O\left(\frac{\log k}{\epsilon}\right)^{\log k}$	$O(n^{1+\frac{1}{k}})$	[15, 10]
$3 + \epsilon$	$k^{\log(3+O(1/\epsilon))}$	$O(n^{1+\frac{1}{k}} \cdot \log \Lambda)$	[3]
$O(c)$	$k^{1+O(1/\log c)}$	$O(n^{1+\frac{1}{k}} \cdot \log \Lambda)$	[3]
$O(k^\epsilon)$	$O_\epsilon(k^{1-\epsilon})$	$O(n^{1+\frac{1}{k}} \cdot \log \Lambda)$	[3]
$2k - 1$	2	$O(n^{1+\frac{1}{k}})$	[21]

1.1 Our Results

We develop a generalization of the TZ-algorithms [21, 22], that achieves (and in some cases improves on) the state-of-the-art for hopsets. This unifies all previous results in a single framework, and greatly simplifies the constructions for hopsets with intermediate stretch (above $1 + \epsilon$ and below $2k - 1$). We also remove all the $\log \Lambda$ factors from the size. This result is summarized in Theorem 1 below.

¹ The aspect ratio is the ratio between the largest distance to the smallest distance in the graph.

In addition, we affirmatively resolve the open problem of [3] mentioned above, by proving that an (α, β) -hopset of size $O(n^{1+1/k})$ must have $\alpha \cdot \beta \geq \Omega(k)$. This lower bound asymptotically matches the upper bound of $(k^\epsilon, O_\epsilon(k^{1-\epsilon}))$ -hopset by [3] for every $0 < \epsilon < 1$.

In the full version of this paper, we also show that whenever our algorithm produces a hopset of size $O(n^{1+1/k})$ with stretch α , it must have a superlinear hopbound of $\beta = \Omega(\frac{1}{\alpha^2} k^{1+1/(2 \log \alpha)})$. This matches the upper bound shown in [3] and here, for all constant α . As our algorithm generalizes all previous constructions, we believe it is an indication that the question whether there exists an $(O(1), O(k))$ -hopset of size $O(n^{1+1/k})$, may have a negative answer.

► **Theorem 1.** *Let $G = (V, E)$ be a weighted undirected graph with n vertices, and fix an integer $k \geq 1$. Then there is an algorithm that can compute each of the following:*

1. *A hopset H of size $O(\log k \cdot n^{1+1/k})$, which is a $(1 + \epsilon, O(\frac{\log k}{\epsilon} \log k))$ -hopset for all $0 < \epsilon < 1$ simultaneously.*
2. *A hopset H of size $O(\log k \cdot n^{1+1/k})$, which is a $(3 + \epsilon, O(k^{\log_2(3 + \frac{16}{\epsilon})}))$ -hopset for all $0 < \epsilon < 1$ simultaneously.*
3. *For any integer $c \geq 1$, an $(8c + 3, O(k^{1 + \frac{2}{\ln c}}))$ -hopset of size $O(c \cdot \log_c k \cdot n^{1+1/k})$.*
4. *For any $0 < \epsilon < 1$, an $(O(e^{2/\epsilon} \cdot k^\epsilon), O(k^{1-\epsilon}))$ -hopset of size $O(n^{1+1/k}/\epsilon^2)$.*
5. *A $(2k - 1, 2)$ -hopset of size $O(k \cdot n^{1+1/k})$.*

1.1.1 Spanners

A closely related concept to hopsets is that of spanners: An (α, β) -spanner of G is a subgraph $H = (V, E'')$ such that for all $u, v \in V$, $d_G(u, v) \leq d_H(u, v) \leq \alpha \cdot d_G(u, v) + \beta$. In the full version of this paper, we describe a unified framework for building spanners, which is a variation of our unified framework for hopsets described here. This unified framework achieves the state-of-the-art results for spanners in essentially all possible values of α .

1.2 Our Techniques

1.2.1 Lower bound

The lower bound on the triple tradeoff between stretch, hopbound and size of (α, β) -hopsets, showing that $\alpha \cdot \beta = \Omega(k)$ whenever the size is $O(n^{1+1/k})$, uses the existence of $n^{1/g}$ -regular graphs with girth g . The basic idea is simple: locally (within distance less than $g/2$) the graph looks like a tree, so when considering short enough paths, of length less than g/α , there are no alternative paths with stretch at most α . This means that any hopset edge (u, v) can only be useful to pairs whose shortest path is “nearby” to u, v . Making this intuition precise, and defining what exactly is “nearby”, requires some careful counting arguments.

We remark that for (α, β) -spanners of size $O(n^{1+1/k})$, there is a better lower bound of $\alpha + \beta \geq \Omega(k)$, which also follows from the family of high girth graphs. (This is because such spanners are in particular $(\alpha + \beta, 0)$ -spanners.) However, this lower bound cannot hold for hopsets, as indicated by the existence of $(O(k^\epsilon), O(k^{1-\epsilon}))$ -hopsets for $\epsilon = 1/2$, say. Indeed, the analysis we use is inherently different, and more intricate, than the one used in the lower bound for spanners.

1.2.2 General algorithm

Before discussing our general algorithm for hopsets, let us review the previous TZ algorithm. Let $G = (V, E)$ be a (possibly weighted) graph with n vertices, and fix an integer parameter k . The algorithms of [21, 22] randomly sample a sequence of sets $V = A_0 \supseteq A_1 \supseteq \dots \supseteq A_F$,

for some F , where each A_{i+1} , $0 \leq i < F$, is sampled by including each vertex from A_i independently with some predefined probability. Then they define for each $v \in V$ its i -th *pivot* $p_i(v)$ as the closest vertex in A_i to v , and the i -th *bunch* as $B_i(v) = \{u \in A_i : d(u, v) < d(v, p_{i+1}(v))\}$. The hopset consists of all edges between each v and some of its bunches.

In [21], the sampling probabilities of each A_{i+1} from A_i were uniform $n^{-1/k}$, i.e., the exponent of $n^{-1/k}$ was linear, so we call this a *linear-TZ*. In this version, each vertex $v \in V$ can connect to vertices in $B_i(v)$ for all $0 \leq i \leq F$. The analysis can give a hopset with $\beta = 2$ and stretch $2k - 1$.

In [22], the sampling probabilities of each A_{i+1} from A_i were roughly $n^{-2^i/k}$, i.e., the exponent of $n^{-2^i/k}$ was exponential in i , so we naturally call this an *exponential-TZ*. As the probabilities are much lower here, the bunches will be larger, so vertices in $A_i \setminus A_{i+1}$ can only connect to their i -th bunch (in order to keep the size under control). This version can provide a near-exact stretch for the hopset.

1.2.3 Our algorithm

In this work, we devise the following generalization of both of these algorithms. Our algorithm expects as a parameter a function $f : \mathbb{N} \rightarrow \mathbb{N}$ that determines, for each level i , the highest bunch-level that vertices in $A_i \setminus A_{i+1}$ will connect to (in the linear-TZ we have $f(i) = F$, while in the exponential-TZ, $f(i) = i$ for all i). This function f implies what should the sampling probability be for each level i , in order to keep the total size of the hopset roughly $O(n^{1+1/k})$. We denote these probabilities by $n^{-\lambda_i/k}$, for parameters $\lambda_0, \lambda_1, \dots, \lambda_{F-1}$. The number of sets F is in turn determined by these λ_i (roughly speaking, it is when we expect A_F to be empty).

As this is a generalization of the algorithms of [21, 22], clearly it may achieve their results. One of our main technical contributions is showing that an *interleaving* of the linear-TZ and exponential-TZ probabilities, yields a hopset with a low hopbound, for any intermediate stretch between $3 + \epsilon$ and k . This means that we divide the integers in $[F]$ to F/t intervals, so that the λ_i 's are the same within each interval, and decays exponentially between intervals. The parameter t controls the stretch.

Our analysis combines ideas from previous works [22, 7, 3], with some novel insights that simplify some of the previously used arguments. In particular, [3] truncated the connections from every vertex in A_i to within a certain range. We show that in our approach, such truncation can be avoided at essentially no cost: this enables our analysis to be scale-free, thereby removing the $\log \Lambda$ factors from the size. In addition, our $(3 + \epsilon, \beta)$ -hopsets combine the best attributes of the hopsets of [7] and [3]: they have no dependence on $\log \Lambda$ and work for all ϵ simultaneously like [7], and have the superior β like [3]. The simplicity of our algorithm has other benefits: for instance, [3] devised two different algorithms, using different tools, for $(O(k^\epsilon), O(k^{1-\epsilon}))$ -hopsets; one for $\epsilon \in (0, 1/2]$ and the other for $\epsilon \in [1/2, 1)$. Our unified algorithm has no need for such separation.

1.3 Organization

In Section 2 we show our lower bound for hopsets. In Section 3 we describe our general algorithm for hopsets, and provide an analysis of its stretch and hopbound in Section 4.

2 Lower Bound for Hopsets

In this section we build a graph G , such that every hopset for G with stretch α and size $O(n^{1+\frac{1}{k}})$ must have a hopbound of at least $\approx \frac{k}{\alpha}$. G has high girth (the size of the smallest simple cycle) and high degree for each vertex, and we prove our lower bound by using counting arguments.

For the construction, we use the following result, which is a well known corollary from a paper by Lubotzky, Phillips and Sarnak [19]:

► **Theorem 2** ([19]). *Given an integer $\gamma \geq 1$, there are infinitely many integers $n \in \mathbb{N}$ such that there exists a $(p+1)$ -regular graph $G = (V, E)$ with $|V| = n$ and girth $\geq \frac{4}{3}\gamma(1 - o(1))$, where $p = D \cdot n^{\frac{1}{\gamma}}$, for some universal constant D .*

Fix $\alpha, \gamma \geq 1$ and a large enough n as above, and let $G = (V, E)$ be the matching $(p+1)$ -regular graph from the theorem. The girth of G is $\geq \frac{4}{3}\gamma(1 - o(1)) > \gamma$. We look at paths in G of distance $\delta := \lfloor \frac{\gamma-1}{\alpha+1} \rfloor$. For a path P , denote by $|P|$ its length. For $u, v \in V$, denote by $P_{u,v}$ a shortest path between them.

► **Lemma 3.** *Suppose $d(u, v) = \delta$. Then for every path P' between u, v such that $|P'| \leq \alpha\delta$, $P_{u,v} \subseteq P'$.*

Proof. If $P_{u,v} \not\subseteq P'$, then $P_{u,v} \cup P'$ contains a simple cycle of length $\leq |P_{u,v}| + |P'| \leq \delta + \alpha\delta = (\alpha+1)\delta \leq \gamma - 1$, in contradiction to the girth of G being $\geq \gamma$. ◀

Lemma 3 implies if $d(u, v) = \delta$, then $P_{u,v}$ is unique. Let $Q_\delta = \{P_{u,v} \mid d(u, v) = \delta\}$.

► **Lemma 4.** $|Q_\delta| \geq \frac{1}{2}np^\delta$.

Proof. Given a vertex $u \in V$, denote its BFS tree, up to the δ 'th level, by T . Since G 's girth is $> (\alpha+1)\delta$, there are no edges between the vertices of T , apart from the edges of T itself. That means that each vertex of T has at least p children at the next level, so we have at least p^δ leaves in T . Each of these leaves is a vertex of distance δ from u , and is connected to u with a δ -path. When summing this quantity over all the vertices $u \in V$, we count every path twice, so we get at least $\frac{1}{2} \sum_{u \in V} p^\delta = \frac{1}{2}np^\delta$ paths of length δ . ◀

We are now ready to prove the main theorem:

► **Theorem 5.** *For every positive integer k , a real number $\alpha > 0$, a constant $C > 0$ and for infinitely many integers n , there exists a graph G with n vertices such that every hopset H for G with size $\leq Cn^{1+\frac{1}{k}}$ and stretch $\leq \alpha$, H has a hopbound $\beta \geq \lfloor \frac{k-2}{\alpha+1} \rfloor$.*

Proof. For α, n and a fixed $\gamma \geq 1$ that will be chosen later, let $G = (V, E)$ be the $(p+1)$ -regular graph from Theorem 2 ($|V| = n$, girth $\geq \gamma$ and $p = D \cdot n^{\frac{1}{\gamma}}$). Define δ, Q_δ the same way as above.

Let H be an (α, β) -hopset for G with size $\leq Cn^{1+\frac{1}{k}}$, where $\beta < \delta$. For $e = (x, y) \in H$, we denote the weight of e , which is defined to be the distance $d(x, y)$, by $w(e)$ ($d(x, y)$ is the distance in the graph G . We omit the subscript from $d_G(u, v)$ for brevity). To formalize our next arguments, we think of a bipartite graph (A, B, \hat{E}) , where $A = Q_\delta$, $B = \{e \in H \mid w(e) \leq \alpha\delta\}$ and $\hat{E} = \{(P, (x, y)) \in A \times B \mid P \cap P_{x,y} \neq \emptyset\}$. We prove the following two properties of this graph ($deg_{\hat{E}}$ denotes the degree of a vertex in this graph):

1. $\forall P \in A \quad deg_{\hat{E}}(P) \geq 1$,
2. $\forall e \in B \quad deg_{\hat{E}}(e) \leq \alpha\delta^2 p^{\delta-1}$.

For (1), we need to show that if $d(u, v) = \delta$, then $\exists(x, y) \in H : P_{u,v} \cap P_{x,y} \neq \emptyset$ and $w(x, y) \leq \alpha\delta$. Let $P \subseteq G \cup H$ be the shortest path from u to v , that has at most β edges, and let \hat{P} be the same path as P , with every H -edge (x, y) replaced by $P_{x,y}$. In \hat{P} , we call the original edges from P *blue* edges, and the other edges *red* edges. By the hopset property: $|\hat{P}| = w(P) = d_{G \cup H}^{(\beta)}(u, v) \leq \alpha \cdot d(u, v) = \alpha\delta$.

From lemma 3, we know that $P_{u,v} \subseteq \hat{P}$, but since \hat{P} contains at most $\beta < \delta = |P_{u,v}|$ *blue* edges, that means that there is a *red* edge in \hat{P} which is in $P_{u,v}$. By the definition of *red* edges, there is some $(x, y) \in H$ such that $P_{x,y}$ contains this *red* edge, therefore $P_{u,v} \cap P_{x,y} \neq \emptyset$. This edge (x, y) is part of P , so we also have $w(x, y) \leq w(P) \leq \alpha\delta$.

For (2), given $(x, y) \in H$ such that $w(x, y) \leq \alpha\delta$, we need to bound the number of pairs $u, v \in V$ such that $d(u, v) = \delta$ and $P_{u,v} \cap P_{x,y} \neq \emptyset$. Let $(a, b) \in P_{x,y}$. Every path of length δ that passes through (a, b) is a concatenation of a path of length i that ends in a , the edge (a, b) and a path of length $\delta - 1 - i$ from b , for some $i \in [0, \delta - 1]$. Fixing i , we can look at the BFS trees T_a, T_b of a, b respectively, up to the i 'th and $(\delta - 1 - i)$ 'th level respectively. Since the degree of any vertex in G is $p + 1$, T_a contains at most p^i leaves, and T_b contains at most $p^{\delta-1-i}$ leaves. Therefore, the number of concatenations of paths as above is bounded by:

$$\sum_{i=0}^{\delta-1} p^i \cdot p^{\delta-1-i} = \sum_{i=0}^{\delta-1} p^{\delta-1} = \delta p^{\delta-1} .$$

Since $P_{x,y}$ contains at most $\alpha\delta$ edges, we get that the number of paths $P_{u,v}$ such that $P_{u,v} \cap P_{x,y} \neq \emptyset$ and $|P_{u,v}| = \delta$, is bounded by $\alpha\delta \cdot \delta p^{\delta-1} = \alpha\delta^2 p^{\delta-1}$. This concludes the proof of (2).

Finally, using the two properties of the bipartite graph, we bound its number of edges from both sides:

$$|\hat{E}| = \sum_{P \in A} \deg_{\hat{E}}(P) \stackrel{(1)}{\geq} |A| = |Q_\delta| \stackrel{\text{lemma 4}}{\geq} \frac{1}{2} np^\delta ,$$

$$|\hat{E}| = \sum_{e \in B} \deg_{\hat{E}}(e) \stackrel{(2)}{\leq} |B| \alpha \delta^2 p^{\delta-1} \leq |H| \alpha \delta^2 p^{\delta-1} .$$

Using these inequalities, we get

$$|H| \geq \frac{1}{2\alpha\delta^2} np = \frac{D}{2\alpha\delta^2} n \cdot n^{\frac{1}{\gamma}} = \frac{D}{2\alpha\delta^2} n^{1+\frac{1}{\gamma}} .$$

Recall that $|H| \leq Cn^{1+\frac{1}{k}}$, so when choosing large enough n , it must be that $k \leq \gamma$. Summarizing our proof so far, we showed that for fixed $\gamma \geq 1$, $\alpha > 0$ and a constant C , there is a graph G such that every (α, β) -hopset H for G , with size $\leq Cn^{1+\frac{1}{k}}$, either satisfies $\beta \geq \delta$, or satisfies $k \leq \gamma$.

Choose $\gamma = k - 1$. Now the matching graph G has the property that every (α, β) -hopset H for G , with size $\leq Cn^{1+\frac{1}{k}}$, must have $\beta \geq \delta$. By δ 's definition:

$$\beta \geq \delta = \lfloor \frac{\gamma - 1}{\alpha + 1} \rfloor = \lfloor \frac{k - 2}{\alpha + 1} \rfloor . \quad \blacktriangleleft$$

3 A Unified Construction of Hopsets

Let $G = (V, E)$ be a weighted undirected graph, and let k be some positive integer.

Our construction of a hopset is a simple generalization of the construction of [21]. We start by constructing a sequence of sets $V = A_0 \supseteq A_1 \supseteq A_2 \supseteq \dots$. The set A_{j+1} is defined by selecting each vertex from A_j independently with some probability that will be defined later.

Given this sequence, define some useful notations:

1. For a vertex $u \in V$, denote by $i(u)$ the *level* of u , which is the only i such that $u \in A_i \setminus A_{i+1}$.
2. The j 'th *pivot* of u , $p_j(u)$, is the vertex of A_j which is the closest to u .
3. The j 'th *bunch* of u is the set $B_j(u) = \{v \in A_j \mid d(u, v) < d(u, p_{j+1}(u))\}$.
(Whenever $A_{j+1} = \emptyset$ or A_{j+1} is undefined, then also $p_{j+1}(u)$ is undefined, and we say that $d(u, p_{j+1}(u)) = \infty$, so $B_j(u) = A_j$).

The hopset construction that relies on [21] adds an hopset edge from each $u \in V$ to the vertices in all of its bunches. Instead, in our case, we add a new parameter to the construction: A non-decreasing function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that $\forall_{i \geq 0} i \leq f(i)$. Now instead of connecting $u \in V$ to *all* of its bunches, we connect it only to bunches $B_j(u)$ with index j between $i(u)$ and $f(i(u))$.

This choice gives us the freedom to change the sampling probability, i.e. the probability that some $v \in A_j$ is chosen to A_{j+1} ; As we will see later, this probability controls the size of the bunches $B_j(u)$, and since u now has less bunches to connect to, they can be larger. In turn, the sampling probability implies how many non-empty sets there will be in the sequence $V = A_0 \supseteq A_1 \supseteq \dots$.

For specifying this dependency between the sampling probability and the parameter f , we use here and throughout this paper, the following notation:

$$f^{-1}(j) = \min\{i \mid j \leq f(i)\}.$$

Given the parameters k, f , we define a sequence $\{\lambda_j\}$ by² $\lambda_j = 1 + \sum_{l < f^{-1}(j)} \lambda_l$.

The sampling probability of a vertex $v \in A_j$ into A_{j+1} is now defined as $n^{-\frac{\lambda_j}{k}}$ (recall that $n = |V|$). We also define $F = \min\{F' \mid \sum_{l < F'} \lambda_l \geq k + 1\}$, and one can simply check that w.h.p. $A_F = \emptyset$.

► **Definition 6.** Given an integer $k \geq 1$ and a non-decreasing function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that $\forall_i f(i) \geq i$, the General Hopset $H(k, f)$, is the hopset:

$$H(k, f) = \bigcup_{u \in V} \bigcup_{j=0}^{F-1} \{(u, p_j(u))\} \cup \bigcup_{u \in V} \bigcup_{j=i(u)}^{f(i(u))} \{(u, v) \mid v \in B_j(u)\}.$$

The following lemma bounds the expected size of our hopset. Its proof appears in the full version of this paper.

► **Lemma 7.** $\mathbb{E}[|H(k, f)|] = O(Fn^{1+\frac{1}{k}})$.

² Note that in the definition of the sequence $\{\lambda_j\}$, no explicit base case was provided (i.e. a definition of λ_0). But, notice that the definition of $\{\lambda_j\}$ actually does contain a definition for λ_0 :

$$\lambda_0 = 1 + \sum_{l < f^{-1}(0)} \lambda_l = 1 + \sum_{l < 0} \lambda_l = 1,$$

where $f^{-1}(0) = 0$ is true by the definition of f^{-1} and the fact that $f(0) \geq 0$.

3.1 Examples

3.1.1 Linear TZ

When choosing $f(j) = k$ for all $j \leq k$, we get $\lambda_j = 1$ for all j (since $f^{-1}(j) = 0$ for all $j \leq k$), and $k + 1 = \sum_{j < F} \lambda_j = F$. The resulting hopset $H(k, f)$ relies on the same construction as in [21], and as observed in [3], it is a $(2k - 1, 2)$ -hopset.

3.1.2 Exponential TZ

Choose $f(j) = j$ for every j . Since $f^{-1}(j) = j$ for every j , we get $\lambda_j = 1 + \sum_{l < j} \lambda_l \Rightarrow \lambda_j = 2^j$ (proof by induction). Also, $k + 1 \approx \sum_{j < F} \lambda_j = 2^F - 1 \Rightarrow F = \lceil \log_2(k + 2) \rceil$. The resulting construction is the same as the emulator from section 4 in [22]. By the analysis of [15, 10], this emulator from [22] is actually a $(1 + \epsilon, O(\frac{\log k}{\epsilon})^{\log k})$ -hopset of size $O(\log k \cdot n^{1 + \frac{1}{k}})$, for every $0 < \epsilon < 1$ simultaneously.

4 Stretch and Hopbound Analysis Method

In this section we show that our general hopset can provide the state-of-the-art results for (α, β) -hopsets, for various regimes of α .

Given a weighted undirected graph G , and given k, f , we add another parameter, which is a sequence of non-negative real numbers: $\{r_i\}_{i=0}^F$. We stress that these parameters only play a part in the analysis.

The following definition of the *score* of a vertex is needed for the lemma that will be proved afterwards.

► **Definition 8.** *Given the function f and the sequence $\{r_i\}$, the **Score** of a vertex $u \in V$ is:*

$$\text{score}(u) = \max\{i > 0 \mid d(u, p_i(u)) > r_i \text{ and } \forall j \in [f^{-1}(i-1), i-1] d(u, p_j(u)) \leq r_j\},$$

where if $p_i(u)$ is not defined (e.g. when $i = F$ and $A_F = \emptyset$), we consider $d(u, p_i(u))$ to be ∞ .

► **Remark 9.** The set in the definition of $\text{score}(u)$ is not empty, so the score of each vertex is well defined and positive. To see this, note that if i is the minimal index such that $d(u, p_i(u)) > r_i$, then $i > 0$ (because $p_0(u) = u$, so $d(u, p_0(u)) = 0 \leq r_0$), $i \leq F$ (because $d(u, p_F(u)) = \infty > r_F$) and also for every $j \in [f^{-1}(i-1), i-1]$, by the minimality of i , $d(u, p_j(u)) \leq r_j$.

Denote $H = H(k, f)$.

► **Lemma 10 (Jumping Lemma).** *Suppose that $\text{score}(u) = i$, then for every $u' \in V$ such that $d(u, u') \leq \frac{r_i - r_{i-1}}{2} - r_{f^{-1}(i-1)}$,*

$$d_{G \cup H}^{(3)}(u, u') \leq 3d(u, u') + 2(r_{i-1} + r_{f^{-1}(i-1)}).$$

Moreover, if also $d(u, u') \geq \frac{2}{t}(r_{i-1} + r_{f^{-1}(i-1)})$ for some $t > 0$, then:

$$d_{G \cup H}^{(3)}(u, u') \leq (t + 3)d(u, u').$$

Proof. Let $u \in V$ be some vertex with $\text{score}(u) = i$ and let $u' \in V$ be some other vertex. We have:

$$d(u', p_{i-1}(u')) \leq d(u', p_{i-1}(u)) \leq d(u', u) + d(u, p_{i-1}(u)) \leq d(u', u) + r_{i-1}. \quad (1)$$

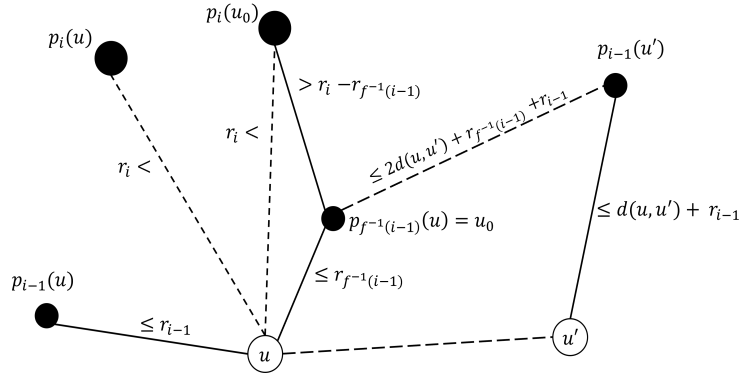
Since $\text{score}(u) = i$, $\forall j \in [f^{-1}(i-1), i-1]$, we have $d(u, p_j(u)) \leq r_j$. In particular, $d(u, p_{f^{-1}(i-1)}(u)) \leq r_{f^{-1}(i-1)}$, and now we can see that:

$$\begin{aligned} d(p_{f^{-1}(i-1)}(u), p_{i-1}(u')) &\leq d(p_{f^{-1}(i-1)}(u), u) + d(u, u') + d(u', p_{i-1}(u')) \\ &\stackrel{(1)}{\leq} r_{f^{-1}(i-1)} + d(u, u') + (d(u, u') + r_{i-1}) \\ &= 2d(u, u') + r_{f^{-1}(i-1)} + r_{i-1}. \end{aligned}$$

For convenience, we denote $u_0 = p_{f^{-1}(i-1)}(u)$, and we also bound the distance $d(u_0, p_i(u_0))$.

$$r_i < d(u, p_i(u)) \leq d(u, p_i(u_0)) \leq d(u, u_0) + d(u_0, p_i(u_0)) \leq r_{f^{-1}(i-1)} + d(u_0, p_i(u_0))$$

$$\Rightarrow d(u_0, p_i(u_0)) > r_i - r_{f^{-1}(i-1)}.$$



■ **Figure 1** The potential path between u and u' . Notice that $d(u, p_{i-1}(u)) \leq r_{i-1}$ and $d(u, p_{f^{-1}(i-1)}(u)) \leq r_{f^{-1}(i-1)}$, since $\text{score}(u) = i$.

Since u_0 is a $f^{-1}(i-1)$ 'th pivot: $i(u_0) \geq f^{-1}(i-1)$, so using the fact that f is non-decreasing: $i-1 \leq f(f^{-1}(i-1)) \leq f(i(u_0))$. Also, since $d(u_0, p_i(u_0)) > r_i - r_{f^{-1}(i-1)} > 0$, it cannot be that $i(u_0) \geq i$ (otherwise $u_0 = p_i(u_0)$, so $d(u_0, p_i(u_0)) = 0$). Then we got that $i-1 \in [i(u_0), f(i(u_0))]$.

Therefore, u_0 is connected to every vertex of $B_{i-1}(u_0)$. Since $p_{i-1}(u') \in A_{i-1}$, a sufficient condition for $p_{i-1}(u')$ to be in $B_{i-1}(u_0)$, which would imply that $(u_0, p_{i-1}(u')) \in H$, is $2d(u, u') + r_{f^{-1}(i-1)} + r_{i-1} \leq r_i - r_{f^{-1}(i-1)}$, i.e.

$$d(u, u') \leq \frac{r_i - r_{i-1}}{2} - r_{f^{-1}(i-1)}.$$

In case that this criteria is satisfied, and we get a 3-hops path from u to u' with weight:

$$\begin{aligned} d_{G \cup H}^{(3)}(u, u') &\leq r_{f^{-1}(i-1)} + (2d(u, u') + r_{f^{-1}(i-1)} + r_{i-1}) + (d(u, u') + r_{i-1}) \\ &= 3d(u, u') + 2(r_{i-1} + r_{f^{-1}(i-1)}). \end{aligned}$$

If also $r_{i-1} + r_{f^{-1}(i-1)} \leq \frac{t}{2}d(u, u')$ (or equivalently $d(u, u') \geq \frac{2}{t}(r_{i-1} + r_{f^{-1}(i-1)})$), then this path is of weight $\leq 3d(u, u') + td(u, u') = (t+3)d(u, u')$. ◀

81:10 A Unified Framework for Hopsets

Given lemma 10, it's best to choose $\{r_i\}$ such that $\frac{2}{t}(r_{i-1} + r_{f^{-1}(i-1)}) \leq \frac{r_i - r_{i-1}}{2} - r_{f^{-1}(i-1)}$, i.e.

$$r_i \geq \left(1 + \frac{4}{t}\right)r_{i-1} + \left(2 + \frac{4}{t}\right)r_{f^{-1}(i-1)} \quad (2)$$

From now on, we assume that $\{r_i\}$ we chose satisfies this inequality. In particular, $\{r_i\}$ is non-decreasing.

Fix $u, v \in V$, and let $u = u_0, u_1, u_2, \dots, u_d = v$ be the shortest path between them.

► **Lemma 11.** *Suppose that $\text{score}(u_j) = i$ and let $l = \max\{l' \geq j \mid d(u_j, u_{l'}) \leq \frac{r_i - r_{i-1}}{2} - r_{f^{-1}(i-1)}\}$. Then if $l < d$, we have:*

1. $d_{G \cup H}^{(4)}(u_j, u_{l+1}) \leq (t+3)d(u_j, u_{l+1})$
2. $d(u_j, u_{l+1}) \geq \frac{4}{t}r_{f^{-1}(i-1)}$

Proof. Denote by W the weight of the edge (u_l, u_{l+1}) . We look at two different cases.

The first case is that $d(u_j, u_l) \geq \frac{2}{t}(r_{i-1} + r_{f^{-1}(i-1)})$. In this case, by lemma 10:

$$\begin{aligned} d_{G \cup H}^{(4)}(u_j, u_{l+1}) &\leq d_{G \cup H}^{(3)}(u_j, u_l) + W \leq (t+3)d(u_j, u_l) + W \\ &< (t+3)(d(u_j, u_l) + W) = (t+3)d(u_j, u_{l+1}) . \end{aligned}$$

The second case is that $d(u_j, u_l) < \frac{2}{t}(r_{i-1} + r_{f^{-1}(i-1)})$. By lemma 10, inequality (2) and l 's definition:

$$\begin{aligned} d_{G \cup H}^{(4)}(u_j, u_{l+1}) &\leq d_{G \cup H}^{(3)}(u_j, u_l) + W \stackrel{10}{\leq} 3d(u_j, u_l) + 2(r_{i-1} + r_{f^{-1}(i-1)}) + W \\ &\stackrel{(2)}{\leq} 3d(u_j, u_l) + t\left(\frac{r_i - r_{i-1}}{2} - r_{f^{-1}(i-1)}\right) + W \\ &\leq 3d(u_j, u_l) + t(d(u_j, u_l) + W) + W \\ &< (t+3)(d(u_j, u_l) + W) = (t+3)d(u_j, u_{l+1}) . \end{aligned}$$

In both cases, we saw that $d(u_j, u_{l+1}) \geq \frac{2}{t}(r_{i-1} + r_{f^{-1}(i-1)})$, so $d(u_j, u_{l+1}) \geq \frac{4}{t}r_{f^{-1}(i-1)}$. ◀

The following theorem presents the size, the stretch and the hopbound for our hopset, $H(k, f)$. It uses lemma 11 repeatedly between every pair of vertices $u, v \in V$. Note that we choose the minimal sequence $\{r_i\}$, for minimizing the hopbound.

► **Theorem 12.** *Fix an integer $k > 0$, a non-decreasing $f : \mathbb{N} \rightarrow \mathbb{N}$ such that $\forall_i f(i) \geq i$, parameters $\{\lambda_j\}$ such that $\forall_j \lambda_j \leq 1 + \sum_{l < f^{-1}(j)} \lambda_l$ and F such that $\sum_{j < F} \lambda_j \geq k + 1$. We can build a $(2t + 3, O(r_F))$ -hopset for an undirected weighted graph G , simultaneously for every $t > 0$, with size $O(Fn^{1+1/k})$, where $\{r_i\}$ satisfies $r_0 = 1$ and $\forall_{i>0} r_i = (1 + \frac{4}{t})r_{i-1} + (2 + \frac{4}{t})r_{f^{-1}(i-1)}$.*

Proof. Given G, k, f , build $H(k, f)$ on G . By lemma 7, this hopset has the wanted size. Fix $u, v \in V$, and let $u = u_0, u_1, u_2, \dots, u_d = v$ be the shortest path between them. We use lemma 11 to find a path between u and v .

Starting with $j = 0$, find $l = \max\{l' \geq j \mid d(u_j, u_{l'}) \leq \frac{r_i - r_{i-1}}{2} - r_{f^{-1}(i-1)}\}$, where $\text{score}(u_j) = i$. If $l = d$, stop the process and denote $v' = u_j$. Otherwise, set $j \leftarrow l + 1$, and continue in the same way.

This process creates a subsequence of u_0, \dots, u_d : $u = v_0, v_1, v_2, \dots, v_b = v'$, such that for every $j < b$ we have $d_{G \cup H}^{(4)}(v_j, v_{j+1}) \leq (t+3)d(v_j, v_{j+1})$ (by lemma 11). For $v' = v_b$ we have $d(v', v) \leq \frac{r_i - r_{i-1}}{2} - r_{f^{-1}(i-1)}$, where $\text{score}(v') = i$. For this last segment, we get from lemma 10 that

$$d_{G \cup H}^{(3)}(v', v) \leq 3d(v', v) + 2(r_{i-1} + r_{f^{-1}(i-1)}) \leq 3d(v', v) + 4r_F .$$

When summing over the entire path, we get:

$$\begin{aligned} d_{G \cup H}^{(4b+3)}(u, v) &\leq \sum_{j=0}^{b-1} (t+3)d(v_j, v_{j+1}) + 3d(v', v) + 4r_F \\ &= (t+3)d(u, v') + 3d(v', v) + 4r_F \leq (t+3)d(u, v) + 4r_F . \end{aligned}$$

To bound b , we notice that by lemma 11, for every $j < b$: $d(v_j, v_{j+1}) \geq \frac{4}{t} r_{F^{-1}(i-1)} \geq \frac{4}{t} r_0$.

So, the number of these “jumps” couldn’t be greater than $\frac{d(u,v)}{\frac{4}{t}r_0} = \frac{t \cdot d(u,v)}{4r_0}$, and we finally got:

$$d_{G \cup H}^{\left(\frac{t \cdot d(u,v)}{r_0} + 3\right)}(u, v) \leq (t+3)d(u, v) + 4r_F .$$

This is true for every sequence $\{r_i\}$ that satisfies inequality (2) (even if it doesn’t satisfy $r_0 = 1$).

Given such sequence $\{r_i\}$, we can define a new sequence as follows:

$$r'_i = \frac{t \cdot d(u, v) \cdot r_i}{4r_F} .$$

This sequence clearly still satisfies (2), so if we use it instead of $\{r_i\}$, we get that for our specific u, v :

$$\begin{aligned} d_{G \cup H}^{\left(\frac{t \cdot d(u,v)}{r'_0} + 3\right)}(u, v) &\leq (t+3)d(u, v) + 4r'_F \Rightarrow \\ \Rightarrow d_{G \cup H}^{(4r_F + 3)}(u, v) &\leq (t+3)d(u, v) + t \cdot d(u, v) = (2t+3)d(u, v) , \end{aligned}$$

i.e. the stretch of this new path is $2t + 3$, and its hopbound is $4r_F + 3$.

Although we chose $\{r'_i\}$ for a specific pair of vertices, this choice of $\{r'_i\}$ **doesn’t change our construction at all**, but only the analysis. So, we proved that for each $u, v \in V$, there is a path between them in $G \cup H$, with stretch $2t + 3$ and hopbound $4r_F + 3$, for our initial choice of $\{r_i\}$. ◀

4.1 Applications

Table 2 demonstrates the different results that can be achieved by substituting different parameters in our construction. The technical computations can be found in the full version of this paper. All of these applications achieve equivalent or even improved results as of [3].

■ **Table 2** The results achieved by substituting different parameters in our construction.

Parameter Choices	Resulting Stretch	Resulting Hopbound	Resulting Hopset Size
$f(i) = i,$ $t = \frac{\epsilon}{2}$	$3 + \epsilon$	$O(k^{\log_2(3 + \frac{16}{\epsilon})})$	$O(\log k \cdot n^{1 + \frac{1}{k}})$
$f(i) = \lfloor \frac{i}{c} \rfloor \cdot c + c - 1,$ $t = 4c$	$8c + 3$	$O(k^{1 + \frac{2}{\ln c}})$	$O(\log_c k \cdot n^{1 + \frac{1}{k}})$
$f(i) = \lfloor \frac{i}{c} \rfloor \cdot c + c - 1,$ $t = 4c, c = \lceil k^\epsilon \rceil$	$O(e^{\frac{2}{\epsilon}} k^\epsilon)$	$O(k^{1-\epsilon})$	$O(n^{1 + \frac{1}{k}} / \epsilon)$



Note that in all 3 resulting hopsets, the hopset size is improved by a $\log \Lambda$ factor in comparison to [3]. Also, for a stretch of $3 + \epsilon$, we get a single hopset with the mentioned properties *simultaneously* for all $\epsilon > 0$. Finally, in comparison to [3], our $(O_\epsilon(k^\epsilon), O(k^{1-\epsilon}))$ -hopset construction enjoys a simpler algorithm and doesn’t require a separation to cases ($\epsilon < \frac{1}{2}$ and $\frac{1}{2} \leq \epsilon$, as in [3]).

References

- 1 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 568–576, 2017. doi:10.1137/1.9781611974782.36.
- 2 Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected shortest paths via low hop emulators. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, page 322?335, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384321.
- 3 Uri Ben-Levy and Merav Parter. New (a, ?) spanners and hopsets. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '20*, page 1695?1714, USA, 2020. Society for Industrial and Applied Mathematics.
- 4 Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 74–83. ACM, 2019. doi:10.1145/3293611.3331633.
- 5 Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000. doi:10.1145/331605.331610.
- 6 Michal Dory and Merav Parter. Exponentially faster shortest paths in the congested clique. In *Proceedings of the 39th Symposium on Principles of Distributed Computing, PODC '20*, page 59?68, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3382734.3405711.
- 7 Michael Elkin, Yuval Gitlitz, and Ofer Neiman. Almost shortest paths with near-additive error in weighted graphs. *CoRR*, abs/1907.11422, 2019. arXiv:1907.11422.
- 8 Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 128–137, 2016. doi:10.1109/FOCS.2016.22.
- 9 Michael Elkin and Ofer Neiman. On efficient distributed construction of near optimal routing schemes. *Distributed Comput.*, 31(2):119–137, 2018. doi:10.1007/s00446-017-0304-4.
- 10 Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and constant-hopbound hopsets in RNC. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019.*, pages 333–341, 2019. doi:10.1145/3323165.3323177.
- 11 Michael Elkin and Ofer Neiman. Centralized and parallel multi-source shortest paths via hopsets and fast matrix multiplication. *CoRR*, abs/2004.07572, 2020. arXiv:2004.07572.
- 12 Michael Elkin and David Peleg. (1+epsilon, beta)-spanner constructions for general graphs. *SIAM J. Comput.*, 33(3):608–631, 2004. doi:10.1137/S0097539701393384.
- 13 Stephan Friedrichs and Christoph Lenzen. Parallel metric tree embedding based on an algebraic view on moore-bellman-ford. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '16*, pages 455–466, New York, NY, USA, 2016. ACM. doi:10.1145/2935764.2935777.
- 14 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. *J. ACM*, 65(6):36:1–36:40, 2018. doi:10.1145/3218657.
- 15 Shang-En Huang and Seth Pettie. Thorup-zwick emulators are universally optimal hopsets. *Information Processing Letters*, 142, April 2017. doi:10.1016/j.ipl.2018.10.001.
- 16 Philip N. Klein and Sairam Subramanian. A randomized parallel algorithm for single-source shortest paths. *J. Algorithms*, 25(2):205–220, 1997. doi:10.1006/jagm.1997.0888.
- 17 Jakub Lacki and Yasamin Nazari. Near-optimal decremental approximate multi-source shortest paths. *CoRR*, abs/2009.08416, 2020. arXiv:2009.08416.

- 18 Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages. *Distributed Comput.*, 32(2):133–157, 2019. doi:10.1007/s00446-018-0326-6.
- 19 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.
- 20 Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 192–201, New York, NY, USA, 2015. ACM. doi:10.1145/2755573.2755574.
- 21 M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. of the 33rd ACM Symp. on Theory of Computing*, pages 183–192, 2001.
- 22 M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proc. of Symp. on Discr. Algorithms*, pages 802–809, 2006.

Data Structures for Node Connectivity Queries

Zeev Nutov   

The Open University of Israel, Ra'anana, Israel

Abstract

Let $\kappa(s, t)$ denote the maximum number of internally disjoint st -paths in an undirected graph G . We consider designing a data structure that includes a list of cuts, and answers the following query: given $s, t \in V$, determine whether $\kappa(s, t) \leq k$, and if so, return a pointer to an st -cut of size $\leq k$ (or to a minimum st -cut) in the list. A trivial data structure that includes a list of $n(n-1)/2$ cuts and requires $\Theta(kn^2)$ space can answer each query in $O(1)$ time. We obtain the following results.

- In the case when G is k -connected, we show that $2n$ cuts suffice, and that these cuts can be partitioned into $2k+1$ laminar families. Thus using space $O(kn)$ we can answer each min-cut query in $O(1)$ time, slightly improving and substantially simplifying the proof of a recent result of Pettie and Yin [18]. We then extend this data structure to subset k -connectivity.
- In the general case we show that $(2k+1)n$ cuts suffice to return an st -cut of size $\leq k$, and a list of size $k(k+2)n$ contains a minimum st -cut for every $s, t \in V$. Combining our subset k -connectivity data structure with the data structure of Hsu and Lu [7] for checking k -connectivity, we give an $O(k^2n)$ space data structure that returns an st -cut of size $\leq k$ in $O(\log k)$ time, while $O(k^3n)$ space enables to return a minimum st -cut.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases node connectivity, minimum cuts, data structure, connectivity queries

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.82

Related Version *Previous Version*: <https://arxiv.org/abs/2110.09102>

1 Introduction

Let $\kappa(s, t) = \kappa_G(s, t)$ denote the maximum number of internally disjoint st -paths in a graph $G = (V, E)$. W.l.o.g. we will assume that G is connected. An **st -cut** is a subset $Q \subseteq V \cup E$ such that $G \setminus Q$ has no st -path. By Menger's Theorem, $\kappa(s, t)$ equals to the minimum size of an st -cut, and there always exists a minimum st -cut that contains no edge except of st . We consider designing a compact data structure that given $s, t \in V$ and $k < n = |V|$ answers the following k -bounded connectivity/cut queries.

$\text{PCON}_k(s, t)$ (partial connectivity query):	Determine whether $\kappa(s, t) \leq k$.
$\text{PCUT}_k(s, t)$ (partial cut query):	If $\kappa(s, t) \leq k$ then return an st -cut of size $\leq k$.
$\text{CON}_k(s, t)$ (connectivity query):	Return $\min\{\kappa(s, t), k+1\}$.
$\text{CUT}_k(s, t)$ (min-cut query):	If $\kappa(s, t) \leq k$ then return a minimum st -cut.

The query $\text{PCUT}_k(s, t)$ requires $\Theta(k)$ time just to write an st -cut. However, by slightly relaxing the definition, we allow the data structure to include a list of cuts, and to return just a pointer to an st -cut of size $\leq k$ in the list. How short can this list be? By choosing a minimum st -cut for each pair $\{s, t\}$, one gets a list of $n(n-1)/2$ cuts. This gives a trivial data structure, that answers both queries in $O(1)$ time, but requires $\Theta(kn^2)$ space – just store the pairwise connectivities in an $n \times n$ matrix, with pointers to the relevant $O(n^2)$ cuts. For edge connectivity, the Gomory-Hu Cut-Tree [5] shows that there exists such a list of $n-1$ cuts that form a laminar family. However, no similar result is known for the node connectivity case considered here.



© Zeev Nutov;

licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 82; pp. 82:1–82:12

Leibniz International Proceedings in Informatics



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

A graph is **k -connected** if $\kappa(s, t) \geq k$ for all $s, t \in V$. Recently, Pettie and Yin [18], and earlier in the 90's Cohen, Di Battista, Kanevsky, and Tamassia [3], considered the above problem in k -connected graphs. Pettie and Yin [18] suggested for $n \geq 4k$ an $O(kn)$ space data structure, that answers $\text{CON}_k(s, t)$ in $O(1)$ time and $\text{CUT}_k(s, t)$ in $O(k)$ time; they showed that it can be constructed in $\tilde{O}(m + \text{poly}(k)n)$ time. The arguments in [18] are complex, and here by a simpler proof we obtain the following improvement on the $\text{CUT}_k(s, t)$ query.

► **Theorem 1.** *For any k -connected graph, there exists an $O(kn)$ space data structure, that includes a list of $2n$ cuts, that answers $\text{CON}_k(s, t)$ and $\text{CUT}_k(s, t)$ queries in $O(1)$ time.*

All our data structures can be constructed in polynomial time; we will not discuss designing efficient construction algorithms here. We note that the Pettie-Yin [18] data structure also includes a list of $O(n)$ cuts, and maybe it can be modified to return a pointer to a minimum st -cut in $O(1)$ time. In any case, our data structure and arguments are simpler, and we sketch the main ideas below.

- We observe that there exists a forest F , such that for any $st \in E$ with $\kappa(s, t) = k$, either one of s, t has degree k (and a minimum st -cut is the set of k edges incident to s or to t), or $st \in F$; this follows from Mader's Critical Cycle Theorem [12]. Thus for pairs $st \in E$ there are at most n relevant cuts, and it is not hard to see that matching between a pair s, t and its cut can be done in $O(1)$ time.
- For $A \subseteq B$ let ∂A denote the set of neighbors of A in G . Let \mathcal{C} be a family of sets obtained by picking for every node v an inclusion minimal set C_v with $v \in C_v$, $|\partial C_v| = k$, and $|C_v| \leq (n - k)/2$, if such a set exists (we show that C_v is unique, if exists). For any $st \notin E$ with $\kappa(s, t) = k$, a minimum st -cut $Q \subset V$ has a connected component of $G \setminus Q$ of size $\leq (n - k)/2$ that contains s or t ; thus $C_s \in \mathcal{C}$ or $C_t \in \mathcal{C}$. To store \mathcal{C} in $O(kn)$ space, we will show that \mathcal{C} can be partitioned into $O(k)$ laminar families (each laminar family can be represented by a tree). To check whether $\kappa(s, t) = k$ we just need to check that $t \notin C_s \cup \partial C_s$ or $s \notin C_t \cup \partial C_t$; this can be done in $O(1)$ time, using a data structure that answers in $O(1)$ time ancestor/descendant queries in trees.

For a set $S \subseteq V$ of terminals we say that a graph is **k - S -connected** if $\kappa(s, t) \geq k$ for all $s, t \in S$. We will extend Theorem 1 to k - S -connected graphs as follows.

► **Theorem 2.** *For any k - S -connected graph with $|S| \geq 3k$, there exists an $O(k|S|)$ space data structure, that includes a list of $3|S|$ cuts, and answers $\text{CON}_k(s, t)$ and $\text{CUT}_k(s, t)$ queries for node pairs in S in $O(1)$ time.*

For arbitrary graphs, a trivial data structure that answers $\text{CON}_k(s, t)$ and $\text{CUT}_k(s, t)$ queries in $O(1)$ time uses $\Theta(kn^2)$ space. Hsu and Lu [7] showed that there exists an auxiliary directed graph $H = (V, F)$ and an ordered partition S_1, S_2, \dots of V , such that:

- Every part S_i has at most $2k - 1$ neighbors in H , and all of them are in $S_{i+1} \cup S_{i+2} \cup \dots$.
- $\kappa(s, t) \geq k + 1$ iff s, t belong to the same part, or $st \in F$, or $ts \in F$.

They also gave a polynomial time algorithm for constructing such H . Augmenting H by a perfect hashing data structure enables to answer " $st \in F$?" queries in $O(1)$ time. Since $|F| = O(kn)$, this gives an $O(kn)$ space¹ data structure that determines whether $\kappa(s, t) \geq k + 1$ in $O(1)$ time. Furthermore, a collection of such data structures for each $k' = 1, \dots, k + 1$ enables to find $\min\{\kappa(s, t), k + 1\}$ in $O(\log k)$ time, using binary search. However, this data

¹ As in previous works, we ignore the unavoidable $O(\log n)$ factor invoked by storing the indexes of nodes, and assume that any basic arithmetic or comparison operation with indexes can be done in $O(1)$ time.

structure alone cannot answer cut queries for pairs that belong to the same part S_i of the partition. Using our data structure for k - S -connectivity from Theorem 2, and a new bound on the number of relevant cuts, we get the following, see also Table 1.

► **Theorem 3.** *There exists an $O(k^2n)$ space data structure that includes a list of $(2k + 1)n$ cuts, that answers $\text{CON}_k(s, t)$ and $\text{PCUT}_k(s, t)$ queries in $O(\log k)$ time; a list of $k(k + 2)n$ cuts and $O(k^3n)$ space allows to answer also $\text{CUT}_k(s, t)$ in $O(\log k)$ time. Furthermore, space $O(k^2n + n^2)$ allows to answer $\text{CON}_k(s, t)$ and $\text{PCUT}_k(s, t)$ in $O(1)$ time, and space $O(k^3n + n^2)$ allows to answer $\text{CON}_k(s, t)$ and $\text{CUT}_k(s, t)$ in $O(1)$ time.*

Theorems 1, 2, and 3 are proved in section 2, 3, and 4, respectively.

■ **Table 1** Summary of the results in Theorem 3. Note that when k is bounded by a constant, the last row data structure has linear space and answers all queries in $O(1)$ time, which is optimal.

list size	space	$\text{PCON}_k(s, t)$	$\text{PCUT}_k(s, t)$	$\text{CON}_k(s, t)$	$\text{CUT}_k(s, t)$	reference
$n(n - 1)/2$	$O(kn^2)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	folklore
-	$O(n^2)$	$O(1)$	-	$O(1)$	-	folklore
-	$O(kn)$	$O(1)$	-	-	-	[7]
-	$O(k^2n)$	$O(1)$	-	$O(\log k)$	-	[7]
$(2k + 1)n$	$O(k^2n + n^2)$	$O(1)$	$O(1)$	$O(1)$	-	this paper
$k(k + 2)n$	$O(k^3n + n^2)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	this paper
$(2k + 1)n$	$O(k^2n)$	$O(\log k)$	$O(\log k)$	$O(\log k)$	-	this paper
$k(k + 2)n$	$O(k^3n)$	$O(\log k)$	$O(\log k)$	$O(\log k)$	$O(\log k)$	this paper

In parallel to our work, Pettie, Saranurak, and Yin [17] gave a randomized $O(kn \log n)$ space data structure that answer $\text{CON}_k(s, t)$ queries in time $O(\log n)$. This data structure extends the data structure of Iszak and Nutov [8], that in turn is based on an idea of Chuzhoy and Khanna [2]. We briefly describe these results. Given a set $S \subseteq V$ of **terminals**, the edges and the nodes in $V \setminus S$ are called **elements**. The **element connectivity** between $s, t \in S$ is the maximum number of pairwise element disjoint st -paths. The Gomory-Hu tree extends to element connectivity (c.f. [19, 1]), and implies an $O(|S|)$ space data structure that answers element connectivity queries between terminals in $O(1)$ time. The data structure of [8] decomposes a node connectivity instance into $O(k^2 \log n)$ element connectivity instances with $\Theta(n/k)$ terminals each; we will give a generalization of this decomposition in Section 5. For any element connectivity instance G_S with terminal set S and $s, t \in S$, $\kappa(s, t)$ is at most the element st -connectivity in G_S , and for at least one instance an equality holds (an instance with $s, t \in S$ and $Q \cap S = \emptyset$ for some minimum st -cut Q). So, to find $\kappa(s, t)$, one has to find the minimum element st -connectivity, among all instances in which s, t are terminals. There are $O(k^2 \log n)$ element connectivity instances, with $\Theta(n/k)$ terminals each, hence the overall number of terminals is $O(k^2 \log n) \cdot (n/k) = O(kn \log n)$. Iszak and Nutov [8] considered designing a labeling scheme², where efficient query time is not required, and used the connectivity classes data structure in each instance; this enables answering $\text{CON}_k(s, t)$ queries in $O(k \log n)$ time. Pettie, Saranurak, and Yin [17] used element connectivity Gomory-Hu trees instead, and also designed a novel data structure that given s, t finds an instance with the minimum element st -connectivity in $O(\log n)$ time. They also

² For additional work on labeling schemes for node-connectivity, see for example, [10, 11, 7].

showed that for large values of k , any data structure for answering node connectivity queries needs at least $\Omega(kn/\log n)$ space, matching up to an $O(\log n)$ factor the $O(kn)$ space of a sparse certificate graph [14].

Let us compare between the [17] $O(kn \log n)$ space data structure that answers $\text{CON}_k(s, t)$ queries in $O(\log n)$ time, to our results. First thing to observe is that the [17] data structure can be augmented by a list of $O(kn \log n)$ cuts to support $\text{CUT}_k(s, t)$ queries. Our cut list has size $O(k^2n)$, which is better when $k < \log n$. Moreover, our query time is $O(\log k)$. For example for $k = \log n$ both data structures have space $O(n \log^2 n)$, but our query time is $O(\log \log n)$ while that of [17] is $O(\log n)$. Note that for $k \leq 4$ are known linear space data structures that can answer cut queries in $O(1)$ time; see [6] and [9] for the cases $k = 3$ and $k = 4$, respectively. Our work, that was done independently from [17] and uses totally different techniques, extend this to any constant k , bridging the gap between the result of [17] and the results known for $k \leq 4$.

2 k -connected graphs (Theorem 1)

In this section let $G = (V, E)$ be a k -connected graph. We first explain how to answer the queries for pairs s, t with $st \in E$. Let K denote the set of nodes of degree k in G . If $s \in K$ then the set of edges incident to s is a minimum st -cut for all t . There are $|K| \leq n$ such minimum cuts. This situation can be recognized in $O(1)$ time, hence we may omit such pairs from our analysis and assume that each of s, t has degree $\geq k + 1$.

We say that $st \in E$ is a **critical edge** if $\kappa(s, t) = k$. Let F be the set of critical edges $st \in E$ such that $s, t \in V \setminus K$. Mader's Critical Cycle Theorem [12] states that any cycle of critical edges contains a node of degree k , hence F is a forest. Thus just specifying the edges in F and a list of $|F| \leq n - |K| - 1$ minimum st -cuts for every $st \in F$, gives an $O(kn)$ space data structure that answers the relevant queries in $O(1)$ time.

Henceforth assume that $s, t \in V \setminus K$ and $st \notin E$. We will show that then there exists a list of $n - |K|$ cuts, such that whenever $\kappa(s, t) = k$, there exists a minimum st -cut in the list. However, this is not enough to answer the relevant queries in $O(1)$ time, since we still need to choose the right minimum cut from the list.

For a node subset $A \subseteq V$ let ∂A denote the set of neighbors of A in G . Let $A^* = V \setminus (A \cup \partial A)$ denote the "node complement" of A . We say that A is: a **tight set** if $|\partial A| = k$ and $A^* \neq \emptyset$, an **st -set** if $s \in A$ and $t \in A^*$, and a **small set** if $|A| \leq \frac{n-k}{2}$. Note that A is tight if and only if ∂A is a minimum cut, and A is a union of some, but not all, connected components of $G \setminus \partial A$. The following statement is a folklore, c.f. [15, 16].

► **Lemma 4.** *Let A, B be tight sets. If the sets $A \cap B^*, B \cap A^*$ are both nonempty then they are both tight. If A, B are small and $A \cap B \neq \emptyset$ then $A \cap B$ is tight.*

Let $R = \{s \in V \setminus K : \text{there exist a small tight set containing } s\}$. For $s \in R$ let C_s be the (unique, by Lemma 4) inclusion minimal small tight set that contains s . Let $\mathcal{C} = \{C_s : s \in R\}$. The following lemma shows that the family $\{\partial C : C \in \mathcal{C}\}$ is a "short" list of $n - |K|$ minimum cuts, that for every $s, t \in V \setminus K$ with $st \notin E$ includes a minimum st -cut.

► **Lemma 5.** *Let $s, t \in V \setminus K$ with $st \notin E$. Then $\kappa(s, t) = k$ if and only if at least one of the following holds: (i) $s \in R$ and C_s is an st -set, or (ii) $t \in R$ and C_t is a ts -set. Consequently, $\kappa(s, t) = k$ if and only if the family $\{\partial C : C \in \mathcal{C}\}$ contains a minimum st -cut.*

Proof. If (i) holds then ∂C_s is a minimum st -cut, while if (ii) holds then ∂C_t is a minimum st -cut. Thus $\kappa(s, t) = k$ if (i) or (ii) holds.

Assume now that $\kappa(s, t) = k$ and we will show that (i) or (ii) holds. Let $Q \subset V \setminus \{s, t\}$ be a minimum st -cut. Then one component A of $G \setminus Q$ contains s and the other B contains t . Since $|A| + |B| \leq n - |Q| = n - k$, one of A, B , say A is small. Thus $s \in R$. Since $C_s \subseteq A$ and since $t \in A^*$, we have $t \in C_s^*$. Consequently, ∂C_s is a minimum st -cut, as required. ◀

Two sets A, B are **laminar** if they are disjoint or one of them contains the other; equivalently, A, B are not laminar if they intersect but none of them contains the other. A set family is laminar if its members are pairwise laminar. A laminar family \mathcal{L} on V can be represented by a rooted tree $T = (\mathcal{L} \cup \{V\}, J)$ and a mapping $\psi : V \rightarrow \mathcal{L} \cup \{V\}$. Here B is a child of A in T if B is a maximal set in $\mathcal{L} \setminus \{A\}$ contained A , and for every $v \in V$, $\psi(v)$ is a minimal set in \mathcal{L} that contains v .

► **Lemma 6.** \mathcal{C} can be partitioned into at most $2k + 1$ laminar families.

Proof. Consider two sets $A = C_a$ and $B = C_b$ that are not laminar. Then $a \notin A \cap B$ since otherwise by Lemma 4 $A \cap B$ is a (small) tight set that contains a , contradicting the minimality of $A = C_a$. By a similar argument, $b \notin A \cap B$. We also cannot have both $a \in A \cap B^*$ and $b \in B \cap A^*$, as then by Lemma 4 $A \cap B^*$ is a tight set that contains a , contradicting the minimality of $A = C_a$. Consequently, $a \in \partial B$ or $b \in \partial A$.

Construct an auxiliary directed graph H on node set R and edges set $\{ab : a \in \partial C_b\}$. The indegree of every node in H is at most k . This implies that every subgraph of the underlying graph of H has a node of degree $2k$. A graph is d -degenerate if every subgraph of it has a node of degree d . It is known that any d -degenerate graph can be colored with $d + 1$ colors, in linear time, see [4, 13]. Hence H is $(2k + 1)$ -colorable. Consequently, we can compute in polynomial time a partition of R into at most $2k + 1$ independent sets. For each independent set R_i , the family $\{C_s : s \in R_i\}$ is laminar. ◀

Our data structure for pairs $s, t \in V \setminus K$ with $st \notin E$ consists of:

- A family \mathcal{T} of at most $2k + 1$ trees, where each tree $T \in \mathcal{T}$ with a mapping $\psi_T : V \rightarrow V(T)$ represents one of the at most $2k + 1$ laminar families of tight sets as in Lemma 6; the total number of edges in all trees in \mathcal{T} is at most $n - |K|$.
- For each tree $T \in \mathcal{T}$, a linear space data structure that answers ancestor/descendant queries in $O(1)$ time. This can be done by assigning to each node of T the in-time and the out-time in a DFS search on T .
- A list $\{\partial C_s : s \in R\}$ of $|R| = n - |K|$ minimum cuts; this can be also encoded by an auxiliary directed graph $H = (V, F)$ with edge set $F = \{ts : t \in \partial C_s\}$. Using perfect hashing data structure we can check whether $ts \in F$ in $O(1)$ time.

For every $s \in S$ let T_s be the (unique) tree in \mathcal{T} where C_s is represented. The next statement, that is a direct consequence of Lemma 5, specifies how we answer the queries.

► **Lemma 7.** Let $s, t \in V \setminus K$ with $st \notin E$.

- (i) If in T_s , $\psi_{T_s}(t)$ is not a descendant of $\psi_{T_s}(s)$ and $t \notin \partial C_s$, then ∂C_s is a minimum st -cut.
- (ii) If in T_t , $\psi_{T_t}(s)$ is not a descendant of $\psi_{T_t}(t)$ and $s \notin \partial C_t$, then ∂C_t is a minimum st -cut. Furthermore, if none of (i),(ii) holds then $\kappa(s, t) \geq k + 1$.

It is easy to see that with appropriate pointers, and using perfect hashing data structure to check adjacency in the auxiliary directed graph H , we get an $O(kn)$ space data structure that checks the two conditions in Lemma 7 in $O(1)$ time. If one of the conditions holds, the data structure return a pointer to one of ∂C_s or ∂C_t . Else, it reports that $\kappa(s, t) \geq k + 1$.

This concludes the proof of Theorem 1.

3 Subset connectivity (Theorem 2)

In this section let $G = (V, E)$ be a k - S -connected graph with $|S| \geq 3k$. For the simpler case $S = V$ we related cuts to node subsets. Unfortunately, for the more general subset k -connectivity case, we need slightly more complex objects than sets, as follows.

► **Definition 8.** An ordered pair $\mathbb{A} = (A, A^+)$ of subsets of a groundset V is called a **bisetet** if $A \subseteq A^+$; A is the **inner set** and A^+ is the **outer set** of \mathbb{A} , and $\partial\mathbb{A} = A^+ \setminus A$ is the **boundary** of \mathbb{A} . $A^* = V \setminus A^+$ is the **co-set** of \mathbb{A} and $\mathbb{A}^* = (A^*, V \setminus A)$ is the **co-bisetet** of \mathbb{A} . We say that \mathbb{A} is an **st-bisetet** if $s \in A$ and $t \in A^*$.

In the case $S = V$, the relevant bisetets were $(A, A^+) = (A, A \cup \partial A)$, where A was a tight set. Here we will say that \mathbb{A} is a **tight bisetet** if $A \cap S \neq \emptyset$, $A^* \cap S \neq \emptyset$, and $|\partial\mathbb{A}| + |\delta(\mathbb{A})| = k$, where $\delta(\mathbb{A})$ is the set of edges in G that go from A to A^* . Note that \mathbb{A} is tight if and only if $\partial\mathbb{A} \cup \delta(\mathbb{A})$ is a minimum st -cut for some $s \in A \cap S$ and $t \in A^* \cap S$. We will consider the family $\mathcal{F} = \{(A \cap S, A^+ \cap S) : \mathbb{A} \text{ is tight}\}$ obtained by projecting the tight bisetets on S . Note that for $\mathbb{A} \in \mathcal{F}$ there might be many tight bisetets in G whose projection on S is \mathbb{A} , and that there always exists at least one such bisetet.

► **Definition 9.** The **intersection** and the **union** of two bisetets \mathbb{A}, \mathbb{B} are the bisetets defined by $\mathbb{A} \cap \mathbb{B} = (A \cap B, A^+ \cap B^+)$ and $\mathbb{A} \cup \mathbb{B} = (A \cup B, A^+ \cup B^+)$. The bisetet $\mathbb{A} \setminus \mathbb{B}$ is defined by $\mathbb{A} \setminus \mathbb{B} = (A \setminus B^+, A^+ \setminus B)$. We say that \mathbb{A}, \mathbb{B} : **intersect** if $A \cap B \neq \emptyset$, **cross** if $A \cap B \neq \emptyset$ and $A^* \cap B^* \neq \emptyset$, and **co-cross** if $A \cap B^* \neq \emptyset$ and $B \cap A^* \neq \emptyset$.

We say that $\mathbb{A} \in \mathcal{F}$ is a **small bisetet** if $|A| \leq \frac{|S|-k}{2}$, and \mathbb{A} is a **large bisetet** otherwise. Clearly, $|\partial\mathbb{A}| \leq k$ for all $\mathbb{A} \in \mathcal{F}$. The family \mathcal{F} has the following properties, c.f. [15, 16].

► **Lemma 10.** The family $\mathcal{F} = \{(A \cap S, A^+ \cap S) : \mathbb{A} \text{ is tight}\}$ has the following properties:

1. \mathcal{F} is **symmetric**: $\mathbb{A}^* \in \mathcal{F}$ whenever $\mathbb{A} \in \mathcal{F}$.
2. \mathcal{F} is **crossing**: $\mathbb{A} \cap \mathbb{B}, \mathbb{A} \cup \mathbb{B} \in \mathcal{F}$ whenever $\mathbb{A}, \mathbb{B} \in \mathcal{F}$ cross.
3. \mathcal{F} is **co-crossing**: $\mathbb{A} \setminus \mathbb{B}, \mathbb{B} \setminus \mathbb{A} \in \mathcal{F}$ whenever $\mathbb{A}, \mathbb{B} \in \mathcal{F}$ co-cross.
4. If $\mathbb{A}, \mathbb{B} \in \mathcal{F}$ are small intersecting bisetets then $\mathbb{A} \cap \mathbb{B}, \mathbb{A} \cup \mathbb{B} \in \mathcal{F}$.

► **Definition 11.** We say that a bisetet \mathbb{B} **contains a bisetet** \mathbb{A} and write $\mathbb{A} \subseteq \mathbb{B}$ if $A \subseteq B$ and $A^+ \subseteq B^+$. \mathbb{A}, \mathbb{B} are **laminar** if one of them contains the other or if $A \cap B = \emptyset$. A bisetet family is laminar if its members are pairwise laminar.

For every $s \in S$ let \mathcal{C}_s be the family of all inclusion minimal bisetets $\mathbb{C} \in \mathcal{F}$ with $s \in C$ and $|C| \leq |C^*|$. Let $\mathcal{C} = \cup_{s \in S} \mathcal{C}_s$. Note that $|\mathcal{C}_s| \leq |S| - 1$ and that \mathcal{C}_s can be computed using $|S| - 1$ min-cut computations. The following observation follows from the symmetry of \mathcal{F} .

► **Lemma 12.** Let $\mathbb{A} \in \mathcal{F}$ be an st -bisetet. If $|A| \leq |A^*|$ then there is an st -bisetet in \mathcal{C}_s , and if $|A^*| \leq |A|$ then there is a ts -bisetet in \mathcal{C}_t . Consequently, for any $s, t \in S$, \mathcal{C} contains an st -bisetet or a ts -bisetet.

The next lemma shows that $|\mathcal{C}_s| \leq 3$ if $|S| \geq 3k$.

► **Lemma 13.** For any $s \in S$, \mathcal{C}_s contains at most one small bisetet and at most $\frac{2(|S|-1)}{|S|-k}$ large bisetets. In particular, if $|S| \geq 3k$ then in \mathcal{C}_s there are at most two large bisetets and $|\mathcal{C}| \leq 3|S|$.

Proof. In \mathcal{C}_s there is at most one small bisetet, by property 4 in Lemma 10. No two large bisetets $\mathbb{A}, \mathbb{B} \in \mathcal{C}_s$ cross, as otherwise $s \in \mathbb{A} \cap \mathbb{B}$ and (by property 2 in Lemma 10) $\mathbb{A} \cap \mathbb{B} \in \mathcal{F}$, contradicting the minimality of \mathbb{A}, \mathbb{B} . Thus the sets in $\{C^* : \mathbb{C} \in \mathcal{C}_s\}$ are pairwise disjoint. Furthermore, $|C^*| \geq |C| \geq \frac{|S|-k}{2}$ for every large bisetet $\mathbb{C} \in \mathcal{C}_s$. This implies that the number of large bisetets in \mathcal{C}_s is at most $\frac{|S|-1}{(|S|-k)/2} \leq \frac{2(3k-1)}{2k} < 3$ if $|S| \geq 3k$. ◀

By a proof identical to that of Lemma 6 we have the following.

► **Lemma 14.** *The family of small bisets in \mathcal{C} can be partitioned in polynomial time into at most $2k + 1$ laminar families.*

Proof. Consider two small bisets $\mathbb{A} \in \mathcal{C}_a$ and $\mathbb{B} \in \mathcal{C}_b$ that are not laminar. Then $a \notin A \cap B$ since otherwise $\mathbb{A} \cap \mathbb{B} \in \mathcal{F}$, contradicting the minimality of \mathbb{A} . Similarly, $b \notin A \cap B$. We also cannot have both $a \in A \cap B^*$ and $b \in B \cap A^*$, as then \mathbb{A}, \mathbb{B} co-cross and thus $\mathbb{A} \setminus \mathbb{B}, \mathbb{B} \setminus \mathbb{A} \in \mathcal{F}$, contradicting the minimality of \mathbb{A}, \mathbb{B} . Consequently, $a \in \partial B$ or $b \in \partial A$. The rest of the proof coincides with that of Lemma 6. ◀

Later, we will prove the following.

► **Lemma 15.** *If $|S| \geq 3k$ then the family of large bisets in \mathcal{C} can be partitioned in polynomial time into at most $3(2k + 1)$ laminar families.*

Lemmas 14 and 15 imply that \mathcal{C} can be partitioned into at most $4(2k + 1)$ laminar families. For our purposes, we just need the family of the inner sets of each family to be laminar. Together with Lemma 13 this implies Theorem 2 in the same way as Lemma 6 implies Theorem 1, except the following minor differences.

- Here we have $4(2k + 1)$ laminar families instead of $2k + 1$ laminar families, and for each $s \in S$ we have by Lemma 13 $|\mathcal{C}_s| \leq 3$ instead of $|\mathcal{C}_s| = 1$.
- For each biset $\mathbb{C} \in \mathcal{C}$, our list of cuts will include a mixed cut $\partial \mathbb{A} \cup \delta(\mathbb{A})$ of some tight biset of G whose projection on S is \mathbb{C} .

These differences affect space and query time only by a small constant factor. Thus all we need is to prove Lemma 15, which we will do in the rest of this section.

► **Lemma 16.** *Let $\mathbb{A} \in \mathcal{C}_a$ and $\mathbb{B} \in \mathcal{C}_b$ be two non-laminar large bisets in \mathcal{C} . If $|S| \geq 3k$ then $a \in \partial \mathbb{B}$ or $b \in \partial \mathbb{A}$, or (see Fig. 1(a)): $a, b \in A \cap B$, $A^* \cap B^* = \emptyset$, and both $A \cap B^*, B \cap A^*$ are non-empty.*

Proof. Assume that $a \notin \partial \mathbb{B}$ and $b \notin \partial \mathbb{A}$. We will show that then the case in Fig. 1(a) holds.

Suppose that $a \in A \cap B$; the analysis of the case $b \in A \cap B$ is similar. Then $A^* \cap B^* = \emptyset$; otherwise, \mathbb{A}, \mathbb{B} cross and (by property 2 in Lemma 10) we get $\mathbb{A} \cap \mathbb{B} \in \mathcal{F}$, contradicting the minimality of \mathbb{A} . Furthermore, if $B \cap A^* = \emptyset$ we get $\frac{|S| - k}{2} < |A| \leq |A^*| = |\partial \mathbb{B} \cap A^*| \leq k$, contradicting that $|S| \geq 3k$. By a similar argument, $A \cap B^* \neq \emptyset$. If $a \in A \cap B$ and $b \in B \cap A^*$ (Fig. 1(b)), then \mathbb{A}, \mathbb{B} co-cross (by property 3 in Lemma 10) and thus $\mathbb{B} \setminus \mathbb{A} \in \mathcal{F}$; this contradicts the minimality of \mathbb{B} .

If none of a, b is in $A \cap B$, then (see Fig. 1(c)) $a \in A \cap B^*$ and $b \in B \cap A^*$. Consequently, \mathbb{A}, \mathbb{B} -co-cross, and thus (by property 3 in Lemma 10) $\mathbb{A} \setminus \mathbb{B}, \mathbb{B} \setminus \mathbb{A} \in \mathcal{F}$, contradicting the minimality of \mathbb{A}, \mathbb{B} .

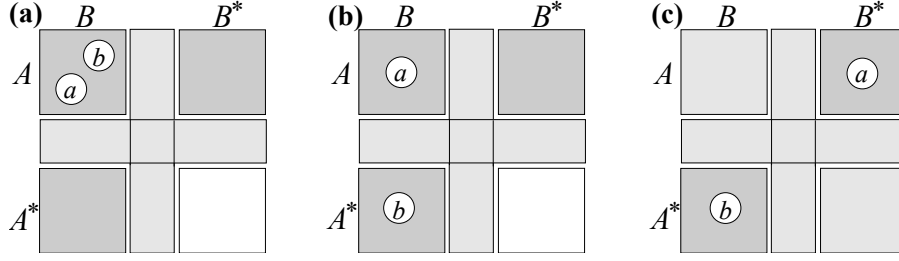
Thus the only possible case is the one in Fig. 1(a), completing the proof of the lemma. ◀

From Lemma 16, by a proof identical to that of Lemma 6 we get the following.

► **Corollary 17.** *The family of large bisets in \mathcal{C} can be partitioned in polynomial time into at most $2k + 1$ parts such that any two bisets $\mathbb{A} \in \mathcal{C}_a$ and $\mathbb{B} \in \mathcal{C}_b$ that belong to the same part \mathcal{P} are either laminar, or have the following property: $a, b \in A \cap B$ and $A^* \cap B^* = \emptyset$.*

Thus the following lemma finishes the proof of Lemma 15, and also of Theorem 2.

► **Lemma 18.** *Let \mathcal{P} be one of the $2k + 1$ parts as in Corollary 17; in particular, if $\mathbb{A}, \mathbb{B} \in \mathcal{P}$ are not laminar then $A \cap B \neq \emptyset$ and $A^* \cap B^* = \emptyset$. Then \mathcal{P} can be partitioned in polynomial time into at most 3 laminar families.*



■ **Figure 1** Illustration to the proof of Lemma 16; dark gray sets are non-empty.

Proof. Let \mathcal{M} be the family of maximal members in \mathcal{P} . We will show later that the set family $\mathcal{H} = \{A^* : A \in \mathcal{M}\}$ has a hitting set U of size $|U| \leq 3$. Now note that:

- For every $v \in S$ the family $\mathcal{P}_v = \{A \in \mathcal{P} : v \in A^*\}$ is laminar, since $v \in A^* \cap B^*$ for any $A, B \in \mathcal{P}_v$, while $A^* \cap B^* = \emptyset$ for any non-laminar $A, B \in \mathcal{P}$.
- Since U is a hitting set of \mathcal{H} , for any $A \in \mathcal{P}$ there is $v \in U \cap A^*$, and then $A \in \mathcal{P}_v$.

Summarizing, each one of the families \mathcal{P}_v is laminar and $\cup_{v \in U} \mathcal{P}_v = \mathcal{P}$. By removing bisets that appear more than once, we get a partition of \mathcal{P} into $|U| \leq 3$ laminar families.

It remains to show that \mathcal{H} has a hitting set of size ≤ 3 . A **fractional hitting set** of \mathcal{H} is a function $h : S \rightarrow [0, 1]$ such that $h(A) = \sum_{v \in A} h(v) \geq 1$ for all $A \in \mathcal{H}$. For $v \in S$ let \mathcal{H}_v be the family of sets in \mathcal{H} that contain v , and let $\mathcal{M}_v = \{A \in \mathcal{M} : A^* \in \mathcal{H}_v\}$. Note that:

- $h(v) = \frac{2}{|S|^{-k+1}}$ for all $v \in S$ is a fractional hitting set of \mathcal{H} and $h(S) = \frac{2|S|}{|S|^{-k+1}} < 3$.
- No two bisets in \mathcal{M}_v intersect and $|\mathcal{H}_v| \leq |\mathcal{M}_v|$. This implies $|\mathcal{H}_v| \leq |\mathcal{M}_v| \leq \frac{2|S|}{|S|^{-k+1}} < 3$, so $|\mathcal{H}_v| \leq 2$ for all $v \in S$.

Since $|\mathcal{H}_v| \leq 2$ for all $v \in S$, computing a minimum hitting set of \mathcal{H} reduces to the minimum Edge-Cover problem, and \mathcal{H} has a hitting set U of size $|U| \leq \frac{4}{3} \cdot h(S)$ ($\frac{4}{3}$ is the integrality gap of the Edge-Cover problem). Since $\frac{4}{3} \cdot h(S) < \frac{4}{3} \cdot 3$, \mathcal{H} has a hitting set U of size $|U| \leq 3$. ◀

4 Arbitrary graphs (Theorem 3)

For the proof of Theorem 3, we will later prove the following.

► **Theorem 19.** *There exists a list of at most $(2k+1)n$ cuts that contains an st -cut of size $\leq k$ for any $s, t \in V$ with $\kappa(s, t) \leq k$.*

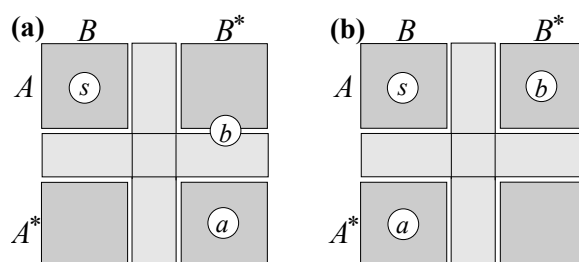
A union of list as in Theorem 19 for every $k' = 1, \dots, k$ is a list that includes a minimum st -cut for any $s, t \in V$ with $\kappa(s, t) \leq k$. The size of this list is $\sum_{k'=1}^k (2k'+1)n = k(k+2)n$. Thus we have the following.

► **Corollary 20.** *There exists a list of at most $k(k+2)n$ cuts that contains a minimum st -cut for any $s, t \in V$ with $\kappa(s, t) \leq k$.*

We can combine these bounds with the trivial data structure. The combined data structure will include a list of cuts, a $V \times V$ matrix, and for each matrix entry (s, t) the number $\min\{\kappa(s, t), k+1\}$ and a pointer to an st -cut in the list. We can answer in $O(1)$ time $\text{PCUT}_k(s, t)$ queries using list of size $O(kn)$ and $\text{CUT}_k(s, t)$ queries using list of size $O(k^2n)$. This proves the second part of Theorem 3.

For the first part of Theorem 3 we combine our bounds with the Hsu-Lu [7] data structure. Recall that this data structure has two ingredients:

- An ordered partition $\mathcal{P} = (S_1, S_2, \dots)$ of V .
- A directed graph $H = (V, F)$ such that every S_i has at most $2k-1$ neighbors in H , and all of them are in $S_{i+1} \cup S_{i+2} \cup \dots$; this implies that $|F| \leq (2k-1)n$.



■ **Figure 2** Illustration to the proof of Lemma 21.

Then $\kappa(s, t) \geq k + 1$ iff s, t belong to the same part, or $st \in F$, or $ts \in F$. Overall, this data structure can be implemented using $O(kn)$ space and answers $\text{PCON}_k(s, t)$ queries in $O(1)$ time.

We augment this data structure by adding to each part $S \in \mathcal{P}$ a data structure for subset k - S -connectivity, with a unified cut list as in Theorem 19; we add Theorem 2 data structure if $|S| \geq 3k$, and the trivial data structure (an $S \times S$ matrix) if $|S| < 3k$.

The dominating space of the combined data structure is $O(k^2n)$, due to storing the Theorem 19 cut list; other parts need substantially lower space. Ignoring the space of the cut list, subset k -connectivity data structures for $S \in \mathcal{P}$ with $|S| \geq 3k$ need total space $\sum_{S \in \mathcal{P}} O(|S|) = O(n)$. For $|S| < 3k$ the trivial data structure has space $O(|S|^2) = O(k|S|)$. It is not hard to see that the worse case is when there are $\Theta(n/k)$ parts in \mathcal{P} of size $\Theta(k)$ each. Thus the total space invoked by parts with $|S| < 3k$ is also $O(n)$. Finally, the space required to store F is $O(|F|) = O(kn)$.

Now we observe that a collection of such data structures for each $k' = 1, \dots, k + 1$ enables to find $\min\{\kappa(s, t), k + 1\}$ in $O(\log k)$ time, using binary search. Once $k' = \kappa(s, t)$ is determined, we can also answer any $\text{CUT}_k(s, t)$ query in $O(1)$ time. The dominating space of this data structure is $O(k^3n)$, due to storing the Corollary 20 cut list.

This concludes the proof of Theorem 2, except that we need to prove Theorem 19, which we will do in the rest of this section.

For a biset \mathbb{A} let $\psi(\mathbb{A}) = |\partial\mathbb{A}| + |\delta(\mathbb{A})|$. Here we will say that \mathbb{A} is an **st -tight biset** if \mathbb{A} is an st -biset and $\psi(\mathbb{A}) = \kappa(s, t)$. Note that then $\partial\mathbb{A} \cup \delta(\mathbb{A})$ is a minimum st -cut, and that for any minimum st -cut Q there exists an st -tight biset \mathbb{A} with $\partial\mathbb{A} \cup \delta(\mathbb{A}) = Q$. It is known that the function ψ satisfies the submodular inequality $\psi(\mathbb{A}) + \psi(\mathbb{B}) \geq \psi(\mathbb{A} \cap \mathbb{B}) + \psi(\mathbb{A} \cup \mathbb{B})$, and (by symmetry) also the co-submodular inequality $\psi(\mathbb{A}) + \psi(\mathbb{B}) \geq \psi(\mathbb{A} \setminus \mathbb{B}) + \psi(\mathbb{B} \setminus \mathbb{A})$.

It is known that if \mathbb{A}, \mathbb{B} are both st -tight then so are $\mathbb{A} \cap \mathbb{B}, \mathbb{A} \cup \mathbb{B}$. Let \mathbb{C}_{st} denote the (unique) inclusion minimal st -tight biset. For $s \in S$ let $T_s = \{t \in V : \kappa(s, t) \leq k, |\mathbb{C}_{st}| \leq |\mathbb{C}_{ts}|\}$. Let \mathcal{C}_s be the family of all inclusion minimal bisets in the family $\{\mathbb{C}_{st} : t \in T_s\}$. Let $\mathcal{C} = \cup_{s \in S} \mathcal{C}_s$. One can verify that for any $s, t \in V$ with $\kappa(s, t) \leq k$, \mathcal{C} contains an st -biset or a ts -biset \mathbb{C} with $\psi(\mathbb{C}) \leq k$. We will show that $|\mathcal{C}_s| \leq 2k + 1$ for all $s \in V$. For that, we need the following lemma.

► **Lemma 21.** *Let $\mathbb{A} = \mathbb{C}_{sa}$ and $\mathbb{B} = \mathbb{C}_{sb}$ be distinct bisets in \mathcal{C}_s . Then $a \in \partial\mathbb{B}$ or $b \in \partial\mathbb{A}$.*

Proof. Suppose to the contrary that $a \notin \partial\mathbb{B}$ and $b \notin \partial\mathbb{A}$. If one of a, b is in $A^* \cap B^*$, say $a \in A^* \cap B^*$ (see Fig. 2(a)), then $\mathbb{A} \cup \mathbb{B}$ is an sa -biset and $\mathbb{A} \cap \mathbb{B}$ is an sb -biset. Thus

$$\kappa(s, a) + \kappa(s, b) = \psi(\mathbb{A}) + \psi(\mathbb{B}) \geq \psi(\mathbb{A} \cap \mathbb{B}) + \psi(\mathbb{A} \cup \mathbb{B}) \geq \kappa(s, b) + \kappa(s, a) .$$

Hence equality holds everywhere, so $\mathbb{A} \cap \mathbb{B}$ is sb -tight. This contradicts the minimality of \mathbb{B} .

Else, $a \in A^* \cap B$ and $b \in B^* \cap A$ (see Fig. 2(b)). Then $A \setminus B$ is a bs -biset and $B \setminus A$ is an as -biset. Thus

$$\kappa(s, a) + \kappa(s, b) = \psi(A) + \psi(B) \geq \psi(A \setminus B) + \psi(B \setminus A) \geq \kappa(b, s) + \kappa(a, s).$$

Hence equality holds everywhere, so $A \setminus B$ is as -tight and $B \setminus A$ is bs -tight. This implies $|C_{sa}| > |C_{bs}|$ and $|C_{sb}| > |C_{as}|$, and we get the contradiction $|C_{sa}| + |C_{sb}| > |C_{bs}| + |C_{as}|$. ◀

► **Lemma 22.** $|\mathcal{C}_s| \leq 2k + 1$ for all $s \in V$.

Proof. The proof is similar to the one of Lemma 6. Construct an auxiliary directed graph H on node set T_s and edge set $\{ab : a \in \partial C_{sb}\}$. Note that if H has no edge between a and b then $C_{sa} = C_{sb}$. The indegree of every node in H is at most k . Thus by the same argument as in Lemma 6 we get that the underlying graph of H is $(2k + 1)$ -colorable, and thus T_s can be partitioned into at most $2k + 1$ independent sets. For each independent set T' , the family $\{C_{st} : t \in T'\}$ consists of a single biset. ◀

This concludes the proof of Theorem 19, and thus also of Theorem 3.

5 Decomposition of node connectivity into element connectivity

Recall that given a set $S \subseteq V$ of terminals, The element connectivity between $s, t \in S$ is the maximum number of pairwise element disjoint st -paths, where elements are the edges and the nodes in $V \setminus S$. Let $\kappa_G^S(s, t)$ denote the st -element-connectivity in G . By Menger's Theorem, $\kappa_G^S(s, t)$ equals the minimum size $|C|$ of a set C of elements with $C \cap S = \emptyset$ such $G \setminus C$ has not st -path. It is easy to see that $\kappa_G^S(s, t) \geq \kappa_G(s, t)$, and that an equality holds iff there exists a minimum st -cut C with $C \cap S = \emptyset$. We thus will consider the following problem: given a family \mathcal{C} of subsets of V , find a “small” family \mathcal{S} of subsets of V such that for every $s, t \in V$ and $C \in \mathcal{C}$ with $s, t \notin C$, there is $S \in \mathcal{S}$ with $s, t \in S$ and $C \cap S = \emptyset$; following [2], we will call such a family \mathcal{C} -resilient. The objective can be also to minimize $\sum_{S \in \mathcal{S}} |S|$. Chuzhoy and Khanna [2] showed that if $\{C \subseteq V : |C| \leq k\}$ is the family of all subsets of size $\leq k$, then there exists a \mathcal{C} -resilient family \mathcal{S} of size $|\mathcal{S}| = O(k^3 \ln n)$. They also gave a randomized polynomial time algorithm for finding such \mathcal{S} . The number of subsets of size $\leq k$ is $\binom{n}{k} \approx n^k$, while the relevant family in our case – as in Corollary 20, has a much smaller size $|\mathcal{C}| \leq k(k + 2)n$. We will consider the case of an arbitrary family $\mathcal{C} \subseteq \{C \subseteq V : |C| \leq k\}$, and prove the following.

► **Lemma 23.** *Let \mathcal{C} be a family of sets of size at most k each on a groundset V of size n . Then there exists a \mathcal{C} -resilient family \mathcal{S} of $O(k^2 \ln(n|\mathcal{C}|))$ subsets of V of size $r = \frac{n-k}{k+1}$ each. Furthermore, assigning to each set in $\mathcal{A} = \{S \subseteq V : |S| = r\}$ probability $\Delta = 1/\binom{n-k-2}{r-2}$ and applying randomized rounding $4 \ln(n|\mathcal{C}|)$ times gives such \mathcal{S} w.h.p.*

Proof. If $n \leq 3k + 1$, then the subsets of V of size 2 is a family as required of size $\frac{3k(3k+1)}{2}$, so assume that $n \geq 3k + 2$.

Let $\mathcal{A} = \{S \subseteq V : |S| = r\}$ and $\mathcal{B} = \{(\{s, t\}, C) : s, t \in V, C \in \mathcal{C}\}$. Define a bipartite graph with sides \mathcal{A}, \mathcal{B} by connecting $S \in \mathcal{A}$ to $(\{s, t\}, C) \in \mathcal{B}$ if $s, t \in S$ and $C \cap S = \emptyset$; in this case we will say that S covers $(\{s, t\}, C)$. This defines an instance of the SET COVER problem, where \mathcal{A} are the sets and \mathcal{B} are the elements. The lemma says that there exists a cover $\mathcal{S} \subseteq \mathcal{A}$ of \mathcal{B} that has size $|\mathcal{S}| = O(k^2 \log(n|\mathcal{C}|))$.

A fractional cover of \mathcal{B} is a function $h : \mathcal{A} \rightarrow [0, 1]$ such that

$$\sum \{h(S) : S \in \mathcal{A} \text{ covers } (\{s, t\}, C)\} \geq 1 \quad \forall (\{s, t\}, C) \in \mathcal{B}.$$

The **value** of a fractional cover h is $\sum_{S \in \mathcal{A}} h(S)$. It is known that if there is a fractional cover of value τ , then there is a cover of size $\tau(1 + \ln |\mathcal{B}|)$. We have $|\mathcal{B}| = \frac{n(n-1)}{2} |\mathcal{C}|$, hence $\ln |\mathcal{B}| \leq 2 \ln(n|\mathcal{C}|) - \ln 2$ and $\lceil 2 \ln |\mathcal{B}| \rceil \leq 4 \ln(n|\mathcal{C}|)$.

Our next goal is to show that there is a fractional cover of value $O(k^2)$. We have $|\mathcal{A}| = \binom{n}{r}$. The number of sets in \mathcal{A} that cover a given member $(\{s, t\}, C) \in \mathcal{B}$ is $\Delta = \binom{n-k-2}{r-2}$, which is the number of choices of a set $S \setminus \{s, t\}$ of size $r-2$ from the set $V \setminus (C \cup \{s, t\})$ of size $n-k-2$. Defining $h(S) = 1/\Delta$ for all $S \in \mathcal{A}$ gives a fractional cover of value $|\mathcal{A}|/\Delta$. Denote $m = n - k$. Then:

$$\frac{|\mathcal{A}|}{\Delta} = \frac{\binom{n}{r}}{\binom{m-2}{r-2}} = \frac{m(m-1)}{r(r-1)} \cdot \frac{n!}{(n-r)!} \cdot \frac{(m-r)!}{m!} \leq \frac{m^2}{(r-1)^2} \prod_{i=1}^r \frac{n-i+1}{m-i+1}.$$

Note that for $1 \leq i \leq r$ we have $\frac{n-i+1}{m-i+1} = 1 + \frac{n-m}{m-i+1} \leq 1 + \frac{k}{n-k-r}$. Let us choose r such that $\frac{k}{n-k-r} = \frac{1}{r}$, so $r = \frac{n-k}{k+1}$; assume that r is an integer, as adjustment to floors and ceilings only affects by a small amount the constant hidden in the $O(\cdot)$ term. Since $(1 + 1/r)^r \leq e$ we obtain

$$\prod_{i=1}^r \frac{n-i+1}{m-i+1} \leq \left(1 + \frac{1}{r}\right)^r \leq e.$$

Since we assume that $n \geq 3k + 2$, we have $\frac{n-k}{k+1} \geq 2$ and thus $\frac{m}{r-1} \leq 2(k+1)$. Consequently, we get that $\frac{|\mathcal{A}|}{\Delta} \cdot (1 + \ln |\mathcal{B}|) = O(k^2 \ln(n|\mathcal{C}|))$. This implies that a standard greedy algorithm for SET COVER, produces the required family of size $O(k^2 \ln n|\mathcal{C}|)$. There is a difficulty to implement this algorithm in time polynomial in n (unless $r = \frac{n-k}{k+1}$ is a constant), since $|\mathcal{A}| = \binom{n}{r}$ may not be polynomial in n . Thus we use a randomized algorithm for SET COVER, by rounding each entry to 1 with probability determined by our fractional cover. It is known that repeating this rounding $2 \lceil \ln |\mathcal{B}| \rceil \leq 4 \ln(n|\mathcal{C}|)$ times gives a cover w.h.p., and clearly its expected size is $2 \lceil \ln |\mathcal{B}| \rceil$ times the value of the fractional hitting set. In our case, $h(S) = 1/\Delta = 1/\binom{n-k-2}{r-2}$ for all $S \in \mathcal{A}$. Thus we just need to assign to each set in \mathcal{A} probability $1/\Delta$, and apply randomized rounding $4 \ln(n|\mathcal{C}|)$ times. \blacktriangleleft

Applying Lemma 23 on the family \mathcal{C} as in as in Corollary 20, that has size $|\mathcal{C}| \leq k(k+2)n$, we get that there exists a \mathcal{C} -resilient family \mathcal{S} of $O(k^2 \ln(n|\mathcal{C}|)) = O(k^2 \ln n)$ subsets of V of size $r = \frac{n-k}{k+1}$ each. On the other hand, if $|\mathcal{C}|$ is the family of all subsets of V of size k , then $|\mathcal{C}| = \binom{n}{k} < (ne/k)^k$ and we get the bound $O(k^2 \ln(n|\mathcal{C}|)) = O(k^3 \ln n)$ of Chuzhoy and Khanna [2].

References

- 1 C. Chekuri, T. Rukkanchanunt, and C. Xu. On element-connectivity preserving graph simplification. In *23rd European Symposium on Algorithms (ESA)*, pages 313–324, 2015.
- 2 J. Chuzhoy and S. Khanna. An $O(k^3 \log n)$ -approximation algorithm for vertex-connectivity survivable network design. *Theory of Computing*, 8(1):401–413, 2012.
- 3 R. F. Cohen, G. Di Battista, A. Kanevsky, and R. Tamassia. Reinventing the wheel: an optimal data structure for connectivity queries. In *25th Symposium on Theory of Computing (STOC)*, pages 194–200, 1993.
- 4 P. Erdős and A. Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Hungarica*, 17(1–2):61–99, 1966.
- 5 R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9, 1961.

- 6 J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Computing*, 2(3):135–158, 1973.
- 7 T-H. Hsu and H-I. Lu. An optimal labeling for node connectivity. In *20th International Symposium Algorithms and Computation (ISAAC)*, pages 303–310, 2009.
- 8 R. Izsak and Z. Nutov. A note on labeling schemes for graph connectivity. *Information Processing Letters*, 112(1-2):39–43, 2012.
- 9 A. Kanevsky, R. Tamassia, G. Di Battista, and J. Chen. On-line maintenance of the four-connected components of a graph. In *32nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 793–801, 1991.
- 10 M. Katz, N. A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004.
- 11 A. Korman. Labeling schemes for vertex connectivity. *ACM Transactions on Algorithms*, 6(2), 2010.
- 12 W. Mader. Ecken vom grad n in minimalen n -fach zusammenhängenden graphen. *Archive der Mathematik*, 23:219–224, 1972.
- 13 D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.
- 14 H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k -connected spanning subgraph of a k -connected graph. *Algorithmica*, 7(5&6):583–596, 1992.
- 15 Z. Nutov. Approximating subset k -connectivity problems. *J. Discrete Algorithms*, 17:51–59, 2012.
- 16 Z. Nutov. Improved approximation algorithms for minimum cost node-connectivity augmentation problems. *Theory of Computing Systems*, 62(3):510–532, 2018.
- 17 S. Pettie, T. Saranurak, and L. Yin. Optimal vertex connectivity oracles. In *54th Symposium on Theory of Computing (STOC)*, pages 151–161, 2022.
- 18 S. Pettie and L. Yin. The structure of minimum vertex cuts. In *48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 105:1–105:20, 2021.
- 19 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer Verlag, Berlin Heidelberg, 2003.

Improved Bounds for Online Balanced Graph Re-Partitioning

Rajmohan Rajaraman ✉

Northeastern University, Boston, MA, USA

Omer Wasim ✉

Northeastern University, Boston, MA, USA

Abstract

We study the online balanced graph re-partitioning problem (OBGR) which was introduced by Avin, Bienkowski, Loukas, Pacut, and Schmid [2] and has recently received significant attention [16, 12, 13, 10, 4] owing to potential applications in large-scale, data-intensive distributed computing. In OBGR, we have a set of ℓ clusters, each with k vertices (representing processes or virtual machines), and an online sequence of communication requests, each represented by a pair of vertices. Any request (u, v) incurs unit communication cost if u and v are located in different clusters (and zero otherwise). Any vertex can be migrated from one cluster to another at a migration cost of $\alpha \geq 1$. We consider the objective of minimizing the total communication and migration cost in the competitive analysis framework. The only known algorithms (which run in exponential time) include an $O(k^2\ell^2)$ competitive [2] and an $O(k\ell 2^{O(k)})$ competitive algorithm [4]. A lower bound of $\Omega(k\ell)$ is known [16]. In an effort to bridge the gap, recent results have considered *beyond worst case* analyses including resource augmentation (with augmented cluster capacity [2, 13, 12]) and restricted request sequences (the *learning* model [13, 12, 16]).

In this paper, we give *deterministic, polynomial-time* algorithms for OBGR, which *mildly* exploit resource augmentation (i.e. augmented cluster capacity of $(1 + \varepsilon)k$ for arbitrary $\varepsilon > 0$). We improve beyond $O(k^2\ell^2)$ -competitiveness (for general ℓ, k) by first giving a simple algorithm with competitive ratio $O(k\ell^2 \log k)$. Our main result is an algorithm with a significantly improved competitive ratio of $O(k\ell \log k)$. At a high level, we achieve this by employing i) an ILP framework to guide the allocation of large components, ii) a simple “any fit” style assignment of small components and iii) a charging argument which allows us to bound the cost of migrations. Like previous work on OBGR, our algorithm and analysis are phase-based, where each phase solves an independent instance of the learning model. Finally, we give an $\Omega(\alpha k \ell \log k)$ lower bound on the total cost incurred by any algorithm for OBGR under the learning model, which quantifies the limitation of a phase-based approach.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, graph partitioning, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.83

1 Introduction

Modern data intensive applications which are distributed across data centers or clusters generate a large amount of network traffic [21, 18, 3]. To enable efficient communication among processes or virtual machines that may be dispersed in these clusters, many distributed systems are increasingly re-configurable and demand-aware [5]. Since inter-cluster communication can incur significant cost due to physical distance and limited bandwidth, clusters may strategically migrate processes to reduce the cost of communication, subject to cluster capacity constraints. The online balanced graph re-partitioning (OBGR) problem, introduced by Avin, Bienkowski, Loukas, Pacut, and Schmid [2], is an algorithmic investigation of trade-offs between migration and inter-cluster communication in an environment where the sequence of communication requests is unknown or hard to predict.



© Rajmohan Rajaraman and Omer Wasim;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 83; pp. 83:1–83:15
Leibniz International Proceedings in Informatics



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

In OGBR, we are given ℓ clusters (representing servers or data centers), each holding at most k vertices (representing processes or virtual machines), and an online sequence of edges (representing communication requests). The algorithm maintains a partition of the vertices among the ℓ clusters so that each set of the partition contains at most k vertices. The *communication cost* of serving a request (u, v) is 0 if u and v are in the same cluster and 1, otherwise. Prior to serving any request, an algorithm has the option of migrating any vertex from one cluster to another for a *migration cost* of $\alpha \in \mathbb{Z}^+$. Given an online sequence σ of requests, the cost incurred by an (online) algorithm \mathcal{A} , denoted by $c(\mathcal{A}, \sigma)$ is the sum of the communication costs and migration costs over σ . Let $OPT(\sigma)$ denote the cost incurred by an optimal offline algorithm, which knows σ in advance. We measure the performance of the algorithm in terms of the (*strict*) competitive ratio which is the minimum value of $\rho > 0$ such that for any input sequence σ and a fixed constant $\tau > 0$ (independent of the length of σ) we have $c(\mathcal{A}, \sigma) \leq \rho \cdot OPT(\sigma) + \tau$. We usually refer to $OPT(\sigma)$ as OPT when σ is clear from context.

The static version of balanced graph partitioning and its variants are well studied [14, 20, 15, 1]. In this problem, given a weighted graph on a set V of n vertices and an integer ℓ , the goal is to partition V into vertex sets V_1, \dots, V_ℓ such that the total weight of edges of the form (u, v) where $u \in V_i, v \in V_j, j \neq i$ is minimized. The problem is NP-hard and even hard to approximate within a finite factor. Note that for $k = 2$, this corresponds to maximum matching and for $\ell = 2$, this reduces to the minimum bisection problem which is already NP-hard [11]. Several approximation and bi-criteria approximation algorithms are known [9, 8, 6, 7] (for a discussion of results, see [2]). Since balanced graph partitioning is NP-hard in the static setting, exponential time competitive algorithms have been considered in the online setting [2, 16, 13]. Note that a balanced partition of the graph induced by the entire request sequence may not necessarily correspond to the optimal offline algorithm's strategy since this strategy overlooks the initial assignment of vertices in clusters (and thus, the migration cost required to mimic a balanced partition), the length of the sequence and its evolution over time. On the other hand, there is an approximation-preserving polynomial time reduction from the static version of OBGR to the offline version of OBGR that repeats the edges of the hard static instance sufficiently many times to derive a hard offline instance. Since the offline problem is unlikely to admit any known polynomial time optimal algorithms, beyond worst-case analysis has been employed to study competitiveness and running times of OBGR. We briefly discuss two such settings in which OBGR has been studied.

Resource Augmentation

In the resource augmented setting, an $(1 + \varepsilon)$ -*augmented* online algorithm is granted $(1 + \varepsilon)k$ capacity on each cluster for some constant $\varepsilon > 0$, and its performance is compared with the optimal offline algorithm with capacity exactly k per cluster. This is similar in vein to the offline bi-criteria versions of the offline balanced graph partitioning problem [6, 7] where the algorithm is required to partition V into ℓ clusters that minimizes the weighted sum of cut edges, such that the number of excess vertices assigned to any cluster is at most δk for some $\delta > 0$. The cost of an algorithm's obtained partition is compared to the cost of an optimal partition of V in which clusters are assigned exactly k vertices. We note that resource augmentation has been studied extensively in online algorithms (e.g., see [17, 24, 25]), and goes back as far as the earliest work on caching [22].

Constrained Input

A special case of OBGR that has been recently considered is the so-called *learning model*, introduced by Henzinger, Neumann, Räcke, and Schmid [12] and studied later in [4, 16]. In this model, the online sequence satisfies the condition that there exists a feasible assignment of vertices to clusters without any inter-cluster requests in the sequence. Thus, upon executing such an assignment of vertices, any algorithm incurs zero cost. In other words, an online algorithm in this model is required to *learn* an optimal partitioning of V into k clusters with no inter-cluster edges. In contrast to the *general model* (i.e. with an arbitrary request sequence), the learning model focuses only on migration costs.

1.1 Related work

OBGR without resource augmentation

In [2], an $O(k^2\ell^2)$ upper bound and an $\Omega(k)$ lower bound are established on the competitive ratio of any deterministic algorithm for OBGR without resource augmentation. The lower bound has been improved to $\Omega(k\ell)$ in recent work by Pacut, Parham, and Schmid [16]. The special cases of $k = 2$ (online re-matching problem) and $k = 3$ have also been studied [2, 16]. In very recent work, Bienkowski, Böhm, Koutecký, Rothvoß, Sgall, and Veselý [4] give an $O(k\ell 2^{O(k)})$ -competitive algorithm for OBGR, which is optimal for constant k .

OBGR with resource augmentation

The $\Omega(k)$ lower bound of [2] holds even when the algorithm is allowed an arbitrary amount of resource augmentation as long as $\ell \geq 2$ and all vertices do not fit into a single cluster. The main result of [2] is an $O(k \log k)$ -competitive deterministic algorithm for OBGR with $(2 + \varepsilon)k$ augmented cluster capacity for $\varepsilon \in (0, 1)$. Very recently, Forner, Räcke, and Schmid [10] give a *polynomial time* deterministic $O(k \log k)$ -competitive algorithm in the same setting.

The learning model

In [16], the authors present a tight $\Theta(k\ell)$ bound for the best deterministic competitive ratio in the learning model without resource augmentation. Moreover, they show that a lower bound of $\Omega(\ell)$ holds even in the $(1 + \varepsilon)$ -augmented setting for $\varepsilon < 1/3$. Henzinger, Neumann and Schmid [13] introduced the learning model of OBGR and give a $O((\ell \log \ell \log n)/\varepsilon)$ -competitive algorithm and a lower bound of $\Omega(1/\varepsilon + \log n)$ assuming $(1 + \varepsilon)k$ augmented capacity for $\varepsilon \in (0, 1/2)$. In more recent work, [12] establishes tight bounds of $\Theta(\log \ell + \log k)$ and $\Theta(\ell \log k)$ on the best competitive ratio of randomized and deterministic algorithms, respectively, for the learning model with resource augmentation.

Summarizing, for deterministic competitive ratios, the best known upper bound for OBGR is $O(k^2\ell^2)$ without resource augmentation and $O(k \log k)$ with $(2 + \varepsilon)$ -augmentation, while the best known lower bound is $\Omega(k\ell)$ without resource augmentation and $\Omega(k + \ell \log k)$ with $(1 + \varepsilon)$ -augmentation for $\varepsilon < 1/3$.

1.2 Our results

In this paper, we give online deterministic $(1 + \varepsilon)$ -augmented algorithms for OBGR in the general model, for an arbitrary constant $\varepsilon > 0$. We first observe that a ρ -competitive algorithm for OBGR in the learning model can be used to get a $\rho k \ell$ -competitive algorithm in the general model. The proof is deferred to Appendix A.

► **Observation 1.** *Any ρ -competitive algorithm for OBGR in the learning model can be transformed to a $O(\rho k\ell)$ -competitive algorithm for OBGR in the general model.*

Using the $(1 + \varepsilon)$ -augmented deterministic $O(\ell \log k)$ -competitive algorithm of [12] for the learning model, Observation 1 immediately yields $(1 + \varepsilon)$ -augmented deterministic $O(k\ell^2 \log k)$ -competitive and randomized $O(k\ell(\log k + \log \ell))$ -competitive algorithms for the general model. The algorithm of [12] for the learning model is quite sophisticated and relies on an intricate analysis. In Section 3, we give an alternative simpler algorithm for the general model referred to as \mathcal{A}_S , which admits a direct analysis and attains the same competitive ratio.

► **Theorem 2.** *There exists a deterministic, polynomial time, $(1 + \varepsilon)$ -augmented $O(k\ell^2 \log k)$ -competitive algorithm for OBGR in the general model, for arbitrary constant $\varepsilon > 0$.*

Our main result, given in Section 4, is a polynomial time *deterministic* $(1 + \varepsilon)$ -augmented $O(k\ell \log k)$ -competitive algorithm \mathcal{A}_G , for constant $\varepsilon > 0$; the competitive ratio nearly matches the lower bound of $\Omega(k\ell)$ without resource augmentation [16]. Under resource augmentation, our algorithm is optimal for constant k while for constant ℓ it is within a $O(\log k)$ factor of the optimal (following from the lower bound of $\Omega(k + \ell \log k)$ in the resource augmented setting). For many applications in which k is usually large (such as distributed communication between nodes placed in cloud servers), our algorithms have near-linear instead of an exponential [4] or quadratic [2] dependence on k in previous work.

► **Theorem 3.** *There exists a deterministic, polynomial time $(1 + \varepsilon)$ -augmented $O(k\ell \log k)$ -competitive algorithm for OBGR in the general mode, for arbitrary constant $\varepsilon > 0$.*

The algorithm of Theorem 3 is a “phase-based” algorithm in which each phase solves OBGR in the learning model. The key component of our proof is an upper bound of $O(\alpha k\ell \log k)$ on the total cost of the algorithm in the learning model, starting from an arbitrary initial assignment of vertices. It is natural to ask whether this bound can be improved since any improvement would also yield an improved competitive ratio for OBGR in the general model. The following lower bound, which can be derived from a lower bound instance of [12], rules this out, thus presenting a limitation of a phase-based analysis approach.

► **Theorem 4.** *For any online deterministic (resp., randomized) algorithm with $(1 + \varepsilon)$ -augmentation for the learning model where $\varepsilon > 0$ is an arbitrary constant, there exists a sequence of requests for which the cost (resp., expected cost) is $\Omega(\alpha k\ell \log k)$.*

1.3 Overview of techniques

We highlight the main techniques we use to get a significantly improved competitive ratio for OBGR in the general model. Our algorithms partition the online sequence of requests into contiguous phases, and keep track of the graph induced by the communication requests within a phase. In particular, the algorithms ensure that during any phase all vertices in a connected component of the graph associated with the phase are assigned to the same cluster. On any request (u, v) where u is in component P_1 and v in P_2 , P_1 and P_2 are merged into P_m and subsequently co-located. Components are classified as small or large based on a threshold size Dk where $D = \Theta(\varepsilon^2)$.

For the algorithm \mathcal{A}_S , if P_m is large, we solve an ILP to guide the assignment of large components. Small components may also need to be reassigned. If P_m is small, P_1 is migrated to P_2 's cluster as long as there is enough space. If that is not possible, small components are

reassigned. We ensure that the maximum assigned volume on any cluster is $(1 + \frac{\varepsilon}{4})k$ after the ILP is solved or small components are reassigned. By definition, large component merges happen only $O(1)$ times while at least $\frac{\varepsilon k}{4}$ total volume of small components is successfully migrated between any two small component reassignments. Using a charging argument, we show that every vertex can be charged at most $O(\ell \log k)$ before an optimal offline algorithm incurs a cost of 1 during that phase, yielding Theorem 2.

The approach for algorithm \mathcal{A}_G is as follows. Each small component assigned to a cluster is allocated a volume which is within a $(1 + \frac{\varepsilon}{4})$ factor of the component size. Once a large component is created during a phase, successive assignments of large components created by any merge are handled by ILP used in \mathcal{A}_S . We note that our ILP is similar to that of [12] and we follow their approach to invoke a result on sensitivity analysis of ILPs [19], which limits the change in assignments when a large component is created. This is not sufficient to establish Theorem 3, however, since small components can be completely displaced leading to high migration cost after every merge. Interestingly, we show that a simple “any fit” strategy for small components coupled with a charging argument is sufficient to bound the total migration cost by $O(k\ell \log k)$.

Finally, to establish the lower bound of Theorem 4, we show that for any competitive algorithm \mathcal{A} there exists a request sequence composed of $\Omega(\log k)$ batches of requests and an initial assignment which is $\Omega(k\ell)$ far apart from \mathcal{A} 's assignment such that \mathcal{A} incurs cost at least $\Omega(\alpha k\ell)$ on every batch.

2 Preliminaries

In this section, we present some definitions and high-level structure of our algorithms, which will be useful throughout the paper. Let $[n]$ denote the set of integers $\{1, 2, \dots, n\}$. Let V denote the set of $n = k\ell$ vertices. Let \mathcal{C} denote the set of ℓ clusters. Each cluster $C \in \mathcal{C}$ is initially assigned exactly k vertices. A request is an unordered pair of vertices (u, v) . A connected component P_i induced by a sequence of requests is the maximal set of vertices such that for any $u \in P_i$ there exists $v \in P_i$ s.t. (u, v) was a request in the sequence. The volume of any component P_i is its size $|P_i|$. Our algorithms maintain a set of connected components $\mathcal{P} = \{P_1, P_2, \dots, P_{|\mathcal{P}|}\}$ where $P_i \subseteq V$ for all i and $\bigcup_{i=1}^{|\mathcal{P}|} P_i = V$. Initially, $\mathcal{P} = \{\{u\} | u \in V\}$ i.e. the set of singleton vertices. We refer to a request (u_t, v_t) with $u_t \in P_1$ and $v_t \in P_2$ as an *inter-cluster request* (between P_1 and P_2) if P_1 and P_2 are assigned to different clusters at the start of time t .

Large and small components

Both our algorithms organize components into classes based on their volumes. A component P is in class i if $|P| \in [(1 + \frac{\varepsilon}{4})^{i-1}, (1 + \frac{\varepsilon}{4})^i)$. A component is small if it belongs to a class i where $i \leq c_s = \lfloor \frac{4}{\varepsilon} \ln(\frac{\varepsilon^2 k}{32}) - 2 \rfloor$ where c_s denotes the number of small component classes. Hence, a component is small if it has volume at most Dk where $D < \frac{\varepsilon^2}{32} < \frac{\varepsilon}{4}$ and large otherwise. Note that the number of large component classes, denoted by c_l satisfies $c_l \leq \frac{4+\varepsilon}{\varepsilon} \ln(\frac{1}{D}) + 2 = O(1)$. A *large component* P is understood to be in (large) component class i if it is in class $i + c_s$. We assume $\varepsilon \geq \frac{4}{k}$. For any cluster C , we use $V(C)$, $V_S(C)$, and $V_L(C)$ to denote the total volume of all, small, and large components, assigned to C , respectively.

Phase-based algorithms

Both our algorithms are phase-based: they divide the sequence of requests into phases, and treat each phase as an independent sequence of requests.

► **Definition 5 (Phase).** *A phase p of a sequence σ of requests is a maximal contiguous subsequence of σ such that there exists a feasible assignment of the set of large components induced by p to clusters in \mathcal{C} satisfying the constraint that the total volume of large components assigned to any cluster is at most $(1 + \frac{\varepsilon}{4})k$.*

A request sequence can be naturally partitioned into consecutive phases. Our algorithms begin a phase by setting \mathcal{P} to the set of singletons and an assignment of vertices to clusters such that every cluster $C \in \mathcal{C}$ is assigned exactly k vertices. For all phases p and all $P_i \in \mathcal{P}$ where \mathcal{P} is the set of components induced by p , vertices in P_i are assigned to the same cluster. Note that OPT increases by 1 per phase. For the sake of exposition, we give our algorithms for the case when $\alpha = 1$. In Appendix B, we show that a simple refinement of our algorithms handles the case when $\alpha > 1$, without asymptotically affecting the competitive ratios.

Merge cases

After any request, (u, v) between components P_1 and P_2 (where w.l.o.g., $|P_1| \leq |P_2|$) which are merged to form P_m , our algorithms consider two *merge cases*: *small*, when P_1, P_2 and P_m are small, and *large* when P_m is large. A merge is viewed as a *deletion* of components P_1, P_2 and an *insertion* of P_m .

3 An $O(k\ell^2 \log k)$ -competitive algorithm

In this section, we present \mathcal{A}_S , an $O(k\ell^2 \log k)$ -competitive algorithm. The algorithmic and analytic techniques developed play a key role in the improved algorithm \mathcal{A}_G of Section 4.

We describe how \mathcal{A}_S executes during any phase. Recall that for any inter-cluster request, our algorithm considers two merge cases. For both the cases, \mathcal{A}_S calls subroutine **Balance-Small** to migrate and re-assign small components. For the large merge case, \mathcal{A}_S calls subroutine **Reassign-Large** to solve an integer linear program (ILP) and guide the placement of large components. The ILP has a constant number of variables and constraints and hence can be solved in constant time. To present the ILP, we first introduce the notion of a signature, which encodes the number of large components of each class assigned to a cluster.

► **Definition 6 (Signature).** *A signature $\tau = (\tau_1, \tau_2, \dots, \tau_{c_l})$ for a cluster $C \in \mathcal{C}$ is a non-negative vector of dimension c_l where τ_i is the number of large components of class i that can be assigned to C such that $Dk \sum_{i=1}^{c_l} (1 + \frac{\varepsilon}{4})^{i-1} \tau_i \leq k$.*

► **Lemma 7 (Upper bound on number of signatures).** *The number of possible signatures for any cluster C is $O((\frac{1}{\varepsilon^2})^{c_l})$.*

Proof. Let τ be a possible signature. Note that $\tau_i \leq \frac{k}{Dk} = O(\frac{1}{\varepsilon^2})$ for all $i \in [c_l]$. Therefore, the total number of different signatures is $O((\frac{1}{\varepsilon^2})^{c_l})$. ◀

3.1 The ILP

We describe the ILP which is agnostic to the assignment of small components. Let $T = \{T_1, T_2, \dots\}$ denote the set of all possible signatures where w.l.o.g., T_1 is the all-zeroes vector. Let T_{ij} denote the j^{th} entry of signature T_i . From Lemma 7, $|T| = O(1)$. For each

signature, T_i let variable $x_i \in [0, \ell]$ denote the number of clusters assigned a signature T_i . Furthermore, let $\kappa_j \in [0, \lceil \frac{\ell}{D} \rceil]$ denote the total number of class j large components. The ILP is as follows.

$$\sum_{i=1}^{|T|} T_{ij} x_i = \kappa_j \text{ for all } j \quad \sum_{i=1}^{|T|} x_i = \ell \quad x_i \in [0, \ell] \text{ for all } i \quad (1)$$

In matrix form, the ILP has $n_r = O(\ln(1/\varepsilon^2)) = O(1)$ rows and $n_c = O(|T|) = O(1)$ columns. Thus, the ILP can be solved in polynomial time. The following lemma shows that the total volume of large components assigned to any cluster never exceeds cluster capacities by more than an $\frac{\varepsilon}{4}$ factor.

► **Lemma 8** (Total volume of large components). *Let τ denote the assigned signature to cluster C according to which large components are assigned to C . Then, $V_L(C) < (1 + \frac{\varepsilon}{4})k$.*

Proof. We note that $V_L(C) < (1 + \frac{\varepsilon}{4})Dk \sum_{i=1}^{c_i} (1 + \frac{\varepsilon}{4})^{i-1} \tau_i \leq (1 + \frac{\varepsilon}{4})k$. ◀

Next, we give subroutines **Balance-Small** and **Reassign-Large**.

■ **Algorithm Balance-Small.**

-
- 1: **for** each cluster $C \in \mathcal{C}$ s.t. $V(C) > (1 + \frac{\varepsilon}{4})k$:
 - 2: **while** $V(C) > (1 + \frac{\varepsilon}{4})k$:
 - 3: Migrate a small component P from C to C_1 where $C_1 \leftarrow \arg \min_{C_2 \in \mathcal{C}} V(C_2)$.
-

■ **Algorithm Reassign-Large.**

-
- 1: Solve ILP (1) to obtain solution x .
 - 2: **if** ILP is infeasible: return NULL.
 - 3: Unmark all clusters $C \in \mathcal{C}$ and all large components in \mathcal{P} .
 - 4: **for** $i \in [|T|]$:
 - 5: **for** $r \in [x_i]$:
 - 6: Assign signature T_i to an unmarked cluster C , and mark C .
 - 7: **for** $j \in [c_j]$:
 - 8: Assign an unmarked large component P of class j to C and mark P .
 - 9: Migrate P , if necessary.
-

If large components are assigned according to the subroutine **Reassign-Large**, then $V_L(C) \leq (1 + \frac{\varepsilon}{4})k$ for all $C \in \mathcal{C}$ which follows from Lemma 8. On the other hand, if $V_L(c) \leq (1 + \frac{\varepsilon}{4})k$ for all $C \in \mathcal{C}$ and **Balance-Small** is run, $V(C) \leq (1 + \frac{\varepsilon}{4})k$ thereafter. The latter follows since $D < \frac{\varepsilon}{4}$ and there always exists a cluster C_1 such that $V(C_1) \leq k$.

3.2 The algorithm

For a request (u_t, v_t) where $u_t \in P_1, v_t \in P_2$, the algorithm \mathcal{A}_S proceeds as follows.

Proof Theorem 2. We bound the total migration cost incurred by the algorithm \mathcal{A}_S during a phase. For the large merge case, the migration cost is bounded by $k\ell$. To pay for this cost, we charge each vertex in P_m a cost at most $\frac{\ell}{D}$. Every vertex can be charged $O(c_\ell)$ times in this manner within any phase, since a component size is bounded by k . For all $k\ell$ vertices, this gives a total charge of $O(\frac{k\ell^2}{D}) = O(k\ell^2)$.

■ **Algorithm \mathcal{A}_S .**

Input: Distinct components P_1 and P_2 in clusters C_1 and C_2 , respectively; $|P_1| \leq |P_2|$

- 1: Merge P_1 and P_2 into P_m and update $\mathcal{P}, \mathcal{P}_S$ and \mathcal{P}_L accordingly.
- 2: **if** $C_1 \neq C_2$:
- 3: **if** P_m is small: ▷ Small merge case
- 4: Assign P_m to C_2 .
- 5: **if** $V(C_2) \leq (1 + \frac{\varepsilon}{2})k$: Migrate all vertices of P_1 from C_1 to C_2 .
- 6: **else**: Run **Balance-Small**.
- 7: **else**: ▷ Large merge Case
- 8: Run **Reassign-Large**.
- 9: **if** **Reassign-Large** returns NULL: Start a new phase.
- 10: **else**: Run **Balance-Small**.

For the small merge case, there are two cases. If $V(C_2) \leq (1 + \frac{\varepsilon}{2})k$, then each vertex in P_1 is charged unit cost. Any vertex can be charged at most $O(\log k)$ in this way since $|P_m| \geq 2|P_1|$ yielding a total charge of $O(k\ell \log k)$. If $V(C_2) > (1 + \frac{\varepsilon}{2})k$ the migration cost incurred due to **Balance-Small** is at most $k\ell$. Let X denote the set of vertices that migrated to C_2 since the last invocation of **Balance-Small**. Then, $|X| > \frac{\varepsilon k}{4}$. Each vertex in X is charged $\frac{4\ell}{\varepsilon}$. Note that any vertex can be included in such a set X only $O(\log k)$ times before it is part of a large component. For all $k\ell$ vertices, this charge sums to $O(\frac{k\ell^2 \log k}{\varepsilon})$. Thus, the total amount charged to all vertices during a phase is $O(k\ell^2 \log k)$, completing the proof of the theorem. ◀

4 An $O(k\ell \log k)$ -competitive algorithm

In this section, we present algorithm \mathcal{A}_G . A major shortcoming of \mathcal{A}_S is that a cost of $\Omega(k\ell^2)$ can be incurred for both small and large merge cases. For a large merge case, \mathcal{A}_G addresses this by ensuring that the total volume $O(k)$ large components migrated is $O(k)$ by employing a sensitivity analysis. The $O(k\ell \log k)$ -competitiveness of \mathcal{A}_G crucially hinges on bounding the migration cost of small components after a large merge case by $O(k)$. To this end, we give a simple “any-fit” assignment procedure for small components. Effectively, the algorithm guarantees that the total migration cost for both merge cases is $O(|P_m|)$, which can be charged to P_m . This yields the desired competitive ratio.

The pseudo code of Algorithm \mathcal{A}_G is given below. The algorithm executes as follows. At any given time, the algorithm maintains the property that the volume assigned to every class i component is given by $(1 + \frac{\varepsilon}{4})^i$. Thus, the total assigned volume for a cluster C overestimates the total volume of components assigned to C by a $(1 + \frac{\varepsilon}{4})$ factor. For the large merge case, an ILP is solved to handle assignment of large components similarly to \mathcal{A}_S . The assignment of large components is completely independent of small components. Thus, the reassignment of large components can displace small components. A displacement of small component P is viewed as a deletion and successive (re)insertion of P . In the next section, we give a procedure to handle the large merge case and show that the total volume of large components migrated is $O(k)$ if P_m is large.

4.1 Handling large components

To handle the large merge case, we use ILP (1). Additionally, we employ a well known bound on the sensitivity of optimal ILP solutions.

■ **Algorithm** \mathcal{A}_G .

Input: Components P_1 and P_2 in clusters C_1, C_2 of class i, j respectively; $i \leq j$.

- 1: Merge P_1 and P_2 into P_m and update $\mathcal{P}, \mathcal{P}_S$ and \mathcal{P}_L respectively.
 - 2: **if** P_m is large ▷ **Large merge case** (see **Section 4.1**)
 - 3: Solve ILP(1).
 - 4: Run algorithm **Assign Signatures** and let $\mathcal{C}' \subseteq \mathcal{C}$ be the set of clusters whose signatures changed.
 - 5: **for** all $C \in \mathcal{C}'$
 - 6: **for** all $P \in \mathcal{P}_S$ assigned to C
 - 7: **if** $U(C) < \lceil |P| \rceil_{(1+\frac{\varepsilon}{4})}$
 - 8: Assign and migrate P to $C_3 \in \mathcal{C}$ where $U(C_3) \geq \lceil |P| \rceil_{(1+\frac{\varepsilon}{4})}$.
 - 9: $U(C_3) \leftarrow U(C_3) - \lceil |P| \rceil_{(1+\frac{\varepsilon}{4})}$.
 - 10: **else**
 - 11: $U(C) \leftarrow U(C) - \lceil |P| \rceil_{(1+\frac{\varepsilon}{4})}$. ▷ The assignment of P remains unchanged
 - 12: **else** ▷ **Small merge case** (see **Section 4.2**)
 - 13: **if** $(1 + \frac{\varepsilon}{4})^j \geq |P_1| + |P_2|$
 - 14: Migrate vertices of P_1 to C_2 .
 - 15: **else**
 - 16: **if** $U(C_2) \geq (1 + \frac{\varepsilon}{4})^m - (1 + \frac{\varepsilon}{4})^j$
 - 17: Migrate vertices of P_1 to C_2 .
 - 18: $U(C_2) \leftarrow U(C_2) - (1 + \frac{\varepsilon}{4})^m + (1 + \frac{\varepsilon}{4})^j$.
 - 19: **else**
 - 20: Migrate vertices of P_m to C_3 where $U(C_3) \geq (1 + \frac{\varepsilon}{4})^m$.
 - 21: $U(C_3) \leftarrow U(C_3) - (1 + \frac{\varepsilon}{4})^m$.
-

► **Theorem 9** (reproduced verbatim from [19]). *Let A be an integral $n_r \times n_c$ matrix, such that each subdeterminant of A is at most Δ in absolute value; let b' and b'' be column n_r -vectors, and let c be a row n_c -vector. Suppose $\max\{cx \mid Ax \leq b' : x \text{ integral}\}$ and $\max\{cx \mid Ax \leq b'' : x \text{ integral}\}$ are finite. Then for each optimum solution z' of the first maximum there exists an optimum solution z'' of the second maximum such that $\|z' - z''\|_\infty \leq n_c \Delta (\|b' - b''\|_\infty + 2)$.*

Following the merge, the RHS vector in our ILP changes by at most 1 in the infinity norm. To bound the sub-determinant, we use the Hadamard inequality to derive that $\Delta \leq n_c^{n_c/2} A_{max}^{n_c/2}$, where A_{max} denotes the maximum entry (in absolute value) of the constraint matrix A . Each entry in the constraint matrix of our ILP has value either 1 or T_{ij} so that $A_{max} \leq \frac{k}{Dk} = O(1/\varepsilon^2)$. As a result, $\Delta = O((|T|/\varepsilon^2)^{|T|})$. Thus, the optimal solution to the ILP changes by $O(|T|\Delta)$ in the infinity norm. Since x has dimension $|T|$ the number of signatures which change between any two optimal solutions is $O(|T|^2\Delta)$.

Assigning signatures to clusters

Let $x = (x_1, \dots, x_{|T|})$ denote the optimal solution obtained after solving the ILP. The procedure **Assign Signatures** greedily assigns signatures to clusters. Following greedy assignment of signatures, large components are migrated between clusters whose assigned signatures changed to reflect new component assignments. The pseudo code is given as follows.

► **Lemma 10.** *The number of clusters whose assigned signatures change whenever a large component is created is $O(|T|^2\Delta) = O(1)$.*

Algorithm Assign Signatures.

```

1: Unmark all clusters  $C \in \mathcal{C}$ .
2:  $\mathcal{C}' \leftarrow \emptyset$ 
3: for  $i = 1$  to  $|T|$ :
4:    $z_i = x_i$ .
5:   while  $z_i \neq 0$ :
6:     if there is an unmarked cluster  $C$  which has assigned signature  $T_i$ 
7:       Mark  $C$ .
8:     else
9:       Pick an arbitrary unmarked cluster  $C$ , assign it signature  $T_i$ .
10:      Mark  $C$  and set  $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{C\}$ .
11:       $z_i \leftarrow z_i - 1$ .
12:  $\mathcal{P}_{\mathcal{C}'} \leftarrow \{P \in \mathcal{P}_L \text{ and } P \text{ is assigned to some } C \in \mathcal{C}'\}$ .
13: for  $C \in \mathcal{C}'$  ▷ Migrate large components to reflect the change in signature.
14:    $\tau \leftarrow$  assigned signature of  $C$ .
15:   for  $i \in [c_i]$ 
16:     for  $j \in [\tau_i]$ 
17:        $P \leftarrow$  class  $i$  component in  $\mathcal{P}_{\mathcal{C}'}$ .
18:       Assign  $P$  to  $C$  and migrate if necessary.
19:    $\mathcal{P}_{\mathcal{C}'} \leftarrow \mathcal{P}_{\mathcal{C}'} \setminus \{P\}$ .
    $U(C) \leftarrow (1 + \varepsilon)k - A_L(C)$ .

```

Proof. The greedy procedure ensures that at most $O(|T|\Delta)$ clusters previously assigned a signature T_i for $i \in [|T|]$ are subsequently assigned a new signature. Thus, at most $O(|T|^2\Delta) = O(1)$ clusters change their assigned signatures. ◀

4.2 Handling small components

In this section, we give a simple procedure to assign small components. This procedure is used for both small and large merge cases. In the latter case, small components may need to be re-assigned due to displacements following a re-assignment of large components. Each small component P of class i is allocated volume exactly $(1 + \frac{\varepsilon}{4})^i$ on a cluster to which it is assigned, i.e. the allocated volume of a component is equal to $\lceil |P| \rceil_{(1 + \frac{\varepsilon}{4})}$ where $\lceil x \rceil_{(1 + \frac{\varepsilon}{4})}$ denotes the value x rounded up to the nearest multiple of $(1 + \frac{\varepsilon}{4})$. We introduce some notation. Let $A_L(C)$ and $A_S(C)$ denote volume allocated to large and small components respectively on a cluster $C \in \mathcal{C}$. Let $\mathcal{P}_S(C), \mathcal{P}_L(C) \subseteq \mathcal{P}_S$ denote the set of small and large components respectively assigned to a cluster C . Note that $A_L(C) = Dk \sum_{i=1}^{c_i} \tau_i (1 + \frac{\varepsilon}{4})^i$ where τ is the signature assigned to C . If \mathcal{P} does not have any large components, then $A_L(C) = 0$ for all $C \in \mathcal{C}$. Moreover, $A_S(C) = \sum_{P \in \mathcal{P}_S(C)} \lceil |P| \rceil_{1 + \frac{\varepsilon}{4}}$. We define the unallocated volume $U(C)$ of cluster $C \in \mathcal{C}$ as $U(C) = (1 + \varepsilon)k - A_L(C) - A_S(C)$.

A small component P of class i which is currently unassigned, is assigned to an arbitrary cluster C whose unallocated volume $U(C)$ is greater than $(1 + \frac{\varepsilon}{4})^i$. Note that such a cluster C must always exist since otherwise this implies that the total volume of components exceeds kl , a contradiction. Below, we outline the assignment of small components.

Small merge case

Consider the small merge case in which components P_1 and P_2 of class i (resp. j) currently assigned to C_1 (resp. C_2) are merged into P_m of class m . W.l.o.g., let $i \leq j$. If $(1 + \frac{\epsilon}{4})^j \geq |P_1| + |P_2|$, vertices of P_1 are migrated to C_2 . In this case, $m = j$. On the other hand, if $m \neq j$ there are two cases to consider. If $U(C_2) \geq (1 + \frac{\epsilon}{4})^m - (1 + \frac{\epsilon}{4})^j$ then vertices of P_1 are migrated to C_2 . Else, vertices in $P_1 \cup P_2$ are migrated to cluster C_3 where $U(C_3) \geq (1 + \frac{\epsilon}{4})^m$. In all cases, P_m is allocated a volume of $(1 + \frac{\epsilon}{4})^m$.

Handling displacements

Consider the large merge case in which re-assignment of large components may displace small components. Each small component P assigned to a cluster C whose signature changes after a large merge is assigned to a cluster C' where $U(C') \geq (1 + \frac{\epsilon}{4})^i$. Since only $O(1)$ clusters change signatures, the total volume of small components displaced is bounded by $O(k)$.

Proof of Theorem 3. The migration cost of large and small merge cases is analyzed separately. For the large merge case, it follows by Lemma 10 that the total volume of large components migrated is $O(k)$, since the assigned signatures change for only $O(1)$ clusters. Let $\mathcal{C}' \subseteq \mathcal{C}$ denote the set of clusters whose signatures changed. The total volume of small components assigned to \mathcal{C}' is bounded by $O(k)$. As a result, the total migration cost to reassign both small and large components is $O(k)$ which is charged uniformly to all vertices in P_m . Since P_m is large, each vertex in P_m is charged $O(1)$. Noting that the number of large component classes, $c_\ell = O(1)$, the total amount charged to all vertices during the time they are part of large components is bounded by $O(k\ell)$.

For the small merge case involving components P_1 and P_2 (assigned to C_1 and C_2 respectively), we consider two types of charges. If $U(C_2)$ is sufficient, vertices of the smaller component P_1 are migrated to C_2 , and the migration cost of $|P_1|$ is charged to vertices in P_1 . Each vertex can be charged $O(\log k)$ many times in this manner before it is part of a large component. For all vertices, this type of charge amounts to $O(k\ell \log k)$. On the other hand, if $U(C_2)$ is insufficient and vertices in $P_1 \cup P_2$ are migrated, the migration cost of $O(|P_m|)$ is charged to all vertices in P_m . However, in this case $m > j$. Since $c_s = O(\log k)$, the total charge of this type for all vertices across the phase is $O(k\ell \log k)$.

As a result, the total migration cost during a phase for both small and large cases during any phase is bounded by $O(k\ell \log k)$. ◀

5 Lower bound for the learning model with arbitrary assignment

In this section, we give a lower bound for any deterministic or randomized algorithm for the learning problem in which the initial assignment of vertices by an offline-optimal algorithm \mathcal{A}_{OPT} and an online algorithm can be arbitrary. Our argument follows an approach implicit in an $\Omega(\log k)$ lower bound established in [12] for randomized OBGR in the learning model.

Let $\Gamma_{\mathcal{A}} = (V_1, V_2, \dots, V_\ell)$, where $V_i \subseteq V$ and $|V_i| = k$ for all $i \in [\ell]$ denote an initial assignment of vertices to clusters that an algorithm \mathcal{A} begins with. The initial assignment of vertices that the algorithm \mathcal{A}_{OPT} begins with is analogously defined and denoted by $\Gamma'_{OPT} = (V'_1, V'_2, \dots, V'_\ell)$. Let $\pi : [\ell] \rightarrow [\ell]$ denote a permutation of integers in $[\ell]$ and Π denote the set of all such permutations. Define $d(\Gamma_{\mathcal{A}}, \Gamma_{OPT}) = \min_{\pi \in \Pi} \sum_{i=1}^{\ell} |V_{\pi(i)} \setminus V'_{\pi(i)}|$ as the initial distance between vertex assignments that \mathcal{A} and \mathcal{A}_{OPT} begin with respectively. Note that $d(\Gamma_{\mathcal{A}}, \Gamma_{OPT}) \leq k\ell$. In the learning problem with arbitrary assignments, the initial distance can be arbitrary. We prove the following result.

► **Theorem 4.** *For any online deterministic (resp., randomized) algorithm with $(1 + \varepsilon)$ -augmentation for the learning model where $\varepsilon > 0$ is an arbitrary constant, there exists a sequence of requests for which the cost (resp., expected cost) is $\Omega(\alpha k \ell \log k)$.*

Proof. Let \mathcal{A} denote an algorithm that begins with an initial assignment $\Gamma_{\mathcal{A}}$. We show that there exists an assignment Γ_{OPT} satisfying $d(\Gamma_{\mathcal{A}}, \Gamma_{OPT}) = \Theta(k\ell)$ such that \mathcal{A} incurs at least $\Omega(k\ell \log k)$ while \mathcal{A}_{OPT} incurs no cost. The idea is to construct a sequence σ composed of batches B_j of requests for $j = \Omega(\log k)$ such that \mathcal{A} incurs cost $\Omega(k\ell)$ on each batch. For the sake of the proof, let k be a power of 2. We assume $\varepsilon < \ell - 1$ is a constant and $\ell \geq 2$.

We give some terminology which will be useful. Let \mathcal{P}_i denote the set of components induced by the set of requests $\cup_{j=1}^i B_j$. Within any batch, we define a saturating sequence of requests between components P_1 and P_2 as a sequence of requests of the form (u, v) where $u \in P_1, v \in P_2$ for vertices u and v which are not currently co-located by \mathcal{A} . By definition a saturating sequence of requests terminates once P_1 and P_2 are co-located by \mathcal{A} . Let $C_0 = \{\{u\} | u \in V\}$ denote the set of singletons before \mathcal{A} services the first request.

For the first batch of requests B_1 , each singleton component $\{u\}$ is paired with another component $\{v\}$ such that u and v are not co-located by \mathcal{A} under the initial assignment $\Gamma_{\mathcal{A}}$. For all such pairs $\{u\}, \{v\}$, B_1 consists of the union of all saturating sequence of requests between $\{u\}$ and $\{v\}$ until they are co-located. If at any point in time while the current batch of requests is being served, \mathcal{A} does not co-locate any pair of components P_1, P_2 , a saturating sequence of requests is issued between P_1 and P_2 . Observe that for \mathcal{A} to be competitive, \mathcal{A} must co-locate all request pairs. Moreover, \mathcal{P}_1 consists of $\frac{k\ell}{2}$ components of size 2.

For any batch B_j for $j > 1$, we proceed similarly. Each component P of size $\frac{k}{2^{j-1}}$ is paired with another component Q such that P and Q are not co-located by \mathcal{A} before any request in batch B_j is issued. Thereafter, for all pairs of components P and Q , a saturating sequence of requests is issued. Once all pairs have been co-located, the next batch of requests B_{j+1} is served.

Note that since requests are issued between only two components of similar size with size less than k at any given time, there exists an assignment $\Gamma_{OPT} = (V'_1, V'_2, \dots, V'_k)$ which satisfies that for any $u, v \in V'_i$ for all $i \in [\ell]$, no request of the form (u, v) was included in σ . Thus, \mathcal{A}_{OPT} incurs zero cost.

On the other hand, the migration cost incurred by \mathcal{A} on any batch of requests B_j is $\Omega(k\ell)$. To this end, note that for all $j \in [\log k]$, \mathcal{P}_{j-1} consists of exactly $\frac{k\ell}{2^{j-1}}$ components, each of size 2^{j-1} . At any point where batch B_j is issued, \mathcal{A} utilizes at least $\frac{k\ell}{(1+\varepsilon)k} = \Omega(\ell)$ clusters to assign components. Thus, there exist $\Omega(\frac{k\ell}{2^{j-1}})$ pairs of components that are not co-located by \mathcal{A} and communication requests during batch B_j necessitate migration of at $\Omega(\frac{k\ell}{2^{j-1}})$ components each of size 2^{j-1} . Thus, the total migration cost incurred by \mathcal{A} to service B_j is $\Omega(\alpha k \ell)$. For all $\Omega(\log k)$ batches, this amounts to $\Omega(\alpha k \ell \log k)$.

A similar approach can be employed to construct a probability distribution over request sequences for which every deterministic algorithm incurs an expected cost of at least $\Omega(\alpha k \ell \log k)$. From Yao's minimax principle [23], this yields a lower bound on the expected cost of any randomized algorithm. The distribution of requests is as follows. As above, the sequence proceeds in batches. The probability distribution for a batch is dependent on the components constructed in the preceding batch. For every batch B_j , two components P and Q of size 2^{j-1} are selected at random. Next, all possible requests are issued between vertices in P (resp. Q) and repeated $\Omega(\alpha)$ times. Then, requests of the form (u, v) where $u \in P, v \in Q$ are issued for all possible u, v and repeated $\Omega(\alpha)$ times. This is repeated for batch B_j until there are no components of size 2^{j-1} . It can be shown that for any batch the expected total cost for any deterministic algorithm is $\Omega(\alpha k \ell)$. Since there are $\Omega(\log k)$ batches, this yields the desired $\Omega(\alpha k \ell \log k)$ lower bound, thus completing the proof. ◀

References

- 1 Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2), April 2009. doi:10.1145/1502793.1502794.
- 2 Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Dynamic balanced graph partitioning. *SIAM Journal on Discrete Mathematics*, 34(3):1791–1812, 2020. doi:10.1137/17M1158513.
- 3 Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, IMC '10*, pages 267–280, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1879141.1879175.
- 4 Marcin Bienkowski, Martin Böhm, Martin Koutecký, Thomas Rothvoß, Jiří Sgall, and Pavel Veselý. Improved analysis of online balanced clustering. In *Approximation and Online Algorithms: 19th International Workshop, WAOA 2021, Lisbon, Portugal, September 6–10, 2021, Revised Selected Papers*, pages 224–233, Berlin, Heidelberg, 2021. Springer-Verlag. doi:10.1007/978-3-030-92702-8_14.
- 5 Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I. Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *SIGCOMM Comput. Commun. Rev.*, 41(4):98–109, August 2011. doi:10.1145/2043164.2018448.
- 6 Guy Even, Joseph (Seffi) Naor, Satish Rao, and Baruch Schieber. Fast approximate graph partitioning algorithms. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '97*, pages 639–648, USA, 1997. Society for Industrial and Applied Mathematics.
- 7 Guy Even, Joseph Seffi Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *J. ACM*, 47(4):585–616, July 2000. doi:10.1145/347476.347478.
- 8 Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Rev.*, 48(1):99–130, January 2006. doi:10.1137/050640904.
- 9 Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. Approximating the minimum bisection size (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, STOC '00*, pages 530–536, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/335305.335370.
- 10 Tobias Forner, Harald Räcke, and Stefan Schmid. Online balanced repartitioning of dynamic communication patterns in polynomial time. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 40–54, 2021. doi:10.1137/1.9781611976489.4.
- 11 M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, STOC '74*, pages 47–63, New York, NY, USA, 1974. Association for Computing Machinery. doi:10.1145/800119.803884.
- 12 Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid. Tight bounds for online graph partitioning. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2799–2818, USA, 2021. Society for Industrial and Applied Mathematics.
- 13 Monika Henzinger, Stefan Neumann, and Stefan Schmid. Efficient distributed workload (re-)embedding. In *Abstracts of the 2019 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '19*, pages 43–44, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3309697.3331503.
- 14 Robert Krauthgamer, Joseph (Seffi) Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '09*, pages 942–949, USA, 2009. Society for Industrial and Applied Mathematics.

- 15 T. Leighton, F. Makedon, and S.G. Tragoudas. Approximation algorithms for vlsi partition problems. In *IEEE International Symposium on Circuits and Systems*, pages 2865–2868 vol.4, 1990. doi:10.1109/ISCAS.1990.112608.
- 16 Maciej Pacut, Mahmoud Parham, and Stefan Schmid. Optimal online balanced partitioning. In *INFOCOM 2021*, 2021.
- 17 Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, STOC '97*, pages 140–149, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258533.258570.
- 18 Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. *SIGCOMM Comput. Commun. Rev.*, 45(4):123–137, August 2015. doi:10.1145/2829988.2787472.
- 19 A. Schrijver. Theory of linear and integer programming. In *Wiley-Interscience series in discrete mathematics and optimization*, 1999.
- 20 Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, September 1997. doi:10.1137/S1064827593255135.
- 21 Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15*, pages 183–197, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2785956.2787508.
- 22 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, February 1985. doi:10.1145/2786.2793.
- 23 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227, 1977. doi:10.1109/SFCS.1977.24.
- 24 N. Young. Thek-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, June 1994. doi:10.1007/BF01189992.
- 25 Neal E. Young. On-line file caching. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '98*, pages 82–86, USA, 1998. Society for Industrial and Applied Mathematics.

A From learning to the general model

► **Observation 1.** *Any ρ -competitive algorithm for OBGR in the learning model can be transformed to a $O(\rho k \ell)$ -competitive algorithm for OBGR in the general model.*

Proof. We give an $\rho k \ell$ -competitive algorithm \mathcal{A} for OBGR in the general model using the ρ -competitive algorithm \mathcal{A}_L as a subroutine. We say an assignment $\Gamma : V \rightarrow \mathcal{C}$ is perfect if every cluster is assigned exactly k vertices. The algorithm partitions the request sequence into phases, and treats each phase as an independent sequence of requests. Here, the definition of a phase is slightly different: phase p of σ is a maximal sub-sequence of requests such that there exists a perfect assignment of vertices which satisfies the property that for all $(u, v) \in p$, $\Gamma(u) = \Gamma(v)$, i.e. u and v are assigned to the same cluster. Before a new phase begins, \mathcal{A} sets \mathcal{P} to the set of singletons and migrates vertices so that every cluster has exactly k vertices. During a phase p , \mathcal{A} simply simulates \mathcal{A}_L ; \mathcal{A}_L starts with the same assignment of vertices as \mathcal{A} at the beginning of p . Let \mathcal{A}_{OPT} denote an offline-optimal algorithm.

It is easy to observe that the cost incurred by \mathcal{A}_{OPT} increases by at least 1 in every phase. We claim that \mathcal{A} incurs a cost no more than $\rho k \ell$. To this end, suppose \mathcal{A} incurred a cost more than $\rho k \ell$. Consider an algorithm which an identical assignment of vertices as \mathcal{A} at

the beginning of phase p and immediately moves to a perfect assignment Γ of vertices such that for any $(u, v) \in p$, $\Gamma(u) = \Gamma(v)$ and incurs no cost thereafter throughout p . The cost of this algorithm is at most $k\ell$ which contradicts that \mathcal{A}_L is ρ -competitive. \blacktriangleleft

B The case of general α

In this section, we show how to adapt our algorithms which were given for $\alpha = 1$ to arbitrary α without a degradation in the asymptotic competitive ratio.

► **Theorem 11.** *Let $\mathcal{A} \in \{\mathcal{A}_S, \mathcal{A}_G\}$ denote a $O(\rho)$ competitive algorithm for OBGR for $\alpha = 1$, where $\rho = \Omega(k\ell \log k)$. Then, \mathcal{A} can be modified to an $O(\rho)$ competitive algorithm \mathcal{A}_M to handle the case of arbitrary α .*

Proof. In the case of arbitrary α , merging of two components is beneficial only when sufficient number of requests have been encountered between them. Let $w(P_i, P_j)$ denote the number of requests of the form (u_t, v_t) between components P_i and P_j where $u_t \in P_i, v_t \in P_j$ during a phase. \mathcal{A}_M initializes a phase by setting \mathcal{P} to the set of singletons and $w(\{u\}, \{v\}) = 0$ for all $u, v \in V$. For components P_i and P_j where w.l.o.g. $|P_i| \leq |P_j|$, \mathcal{A}_M merges them into P_m when $w(P_i, P_j) \geq \alpha|P_i|$. For every component $P_r \neq P_i, P_j$, $w(P_m, P_r)$ is set to $w(P_i, P_r) + w(P_j, P_r)$. Due to this reason, it is possible P_r may become eligible to be merged with P_m . A request (u_t, v_t) is *special* if it leads to one or more component merges.

During any phase, \mathcal{A}_M works as follows: on any request (u_t, v_t) between components P_i and P_j it first increments $w(P_i, P_j)$. Next, it determines whether the request is special. If it is special, \mathcal{A}_M simulates \mathcal{A} on this request. Note that if P_i and P_j are in the same cluster, then nothing needs to be done besides updating data structures and merging P_i and P_j into P_m . However, if this makes a component P_r eligible to be merged with P_m , \mathcal{A}_M creates an artificial request (u^A, v^A) where $u^A \in P_m, v^A \in P_r$ and simulates the action of \mathcal{A} on (u^A, v^A) . Recursive component merges are handled similarly. A phase of \mathcal{A}_M ends whenever a phase of \mathcal{A} ends. Note that requests to \mathcal{A} only consist of special and artificial requests.

We bound the total communication and migration cost incurred by \mathcal{A}_M during a phase. Since \mathcal{A} incurs a cost of $O(\rho)$ per phase, the migration cost of \mathcal{A}_M is bounded by $O(\alpha\rho)$. We claim the communication cost per phase of \mathcal{A}_M is $O(\alpha k\ell \log k)$. For this purpose, consider charging any vertex in a small component P_i a cost of α whenever P_i is merged with P_j . This is sufficient to bound the total communication cost, which is $\alpha|P_i|$ incurred due to communication between vertices in P_i and P_j . Thus, every vertex is charged $O(\alpha \log k)$ per phase yielding a total communication cost of $O(\alpha k\ell \log k)$.

To lower bound the cost of an optimal offline algorithm during the phase, note that either it migrated a vertex or not. If a vertex was migrated during the phase, then $OPT \geq \alpha$. On the other hand, if no vertex was migrated, a communication cost of at least α must have been incurred. To see why, note that at the termination of the phase, the ILP 1 solved by \mathcal{A} determines that no feasible solution exists. Each edge in the graph that \mathcal{A} maintains during the phase corresponds to at least α paid communication requests handled by \mathcal{A}_M . Thus, for both cases $OPT \geq \alpha$ per phase.

This yields $O(\rho + k\ell \log k)$ competitiveness. Since $\rho = \Omega(k\ell \log k)$, the theorem follows. \blacktriangleleft

An Empirical Evaluation of k -Means Coresets

Chris Schwiegelshohn ✉

Department of Computer Science, Aarhus University, Denmark

Omar Ali Sheikh-Omar ✉ 

Department of Computer Science, Aarhus University, Denmark

Abstract

Coresets are among the most popular paradigms for summarizing data. In particular, there exist many high performance coresets for clustering problems such as k -means in both theory and practice. Curiously, there exists no work on comparing the quality of available k -means coresets.

In this paper we perform such an evaluation. There currently is no algorithm known to measure the distortion of a candidate coreset. We provide some evidence as to why this might be computationally difficult. To complement this, we propose a benchmark for which we argue that computing coresets is challenging and which also allows us an easy (heuristic) evaluation of coresets. Using this benchmark and real-world data sets, we conduct an exhaustive evaluation of the most commonly used coreset algorithms from theory and practice.

2012 ACM Subject Classification Theory of computation → Data compression; Information systems → Clustering

Keywords and phrases coresets, k -means coresets, evaluation, benchmark

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.84

Related Version *Full Version*: <https://arxiv.org/pdf/2207.00966>

Supplementary Material *Software (Source Code)*: <https://github.com/sheikhomar/eval-k-means-linebreak> coresets, archived at [swh:1:dir:53066aa034ea87cdf2fd2f5cb2077400aaf341c3](https://swh.1:dir:53066aa034ea87cdf2fd2f5cb2077400aaf341c3)

Funding *Chris Schwiegelshohn*: Independent Research Fund Denmark (DRF) Sapere Aude Research Leader grant No 1051-00106B.

Omar Ali Sheikh-Omar: Innovation Fund Denmark under grant agreement No 0153-00233A.

1 Introduction

The design and analysis of scalable algorithms has become an important research area over the past two decades. This is particularly important in data analysis, where even polynomial running time might not be enough to handle proverbial *big data* sets. One of the main approaches to deal with the scalability issue is to compress or sketch large data sets into smaller, more manageable ones. The aim of such compression methods is to preserve the properties of the original data, up to some small error, while significantly reducing the number of data points.

Among the most popular and successful paradigms in this line of research are *coresets* [40]. Informally, given a data set A , a coreset $\Omega \subset A$ with respect to a given set of queries Q and query function $f : A \times Q \rightarrow \mathbb{R}_{\geq 0}$ approximates the behaviour of A for all queries up to some multiplicative distortion D via $\sup_{q \in Q} \max \left(\frac{f(\Omega, q)}{f(A, q)}, \frac{f(A, q)}{f(\Omega, q)} \right) \leq D$. Coresets have been applied to a number of problems such as computational geometry [2, 9], linear algebra [30, 34], and machine learning [36, 41]. But the by far most intensively studied and arguably most successful applications of the coreset framework is the k -clustering problem.



© Chris Schwiegelshohn and Omar Ali Sheikh-Omar;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 84; pp. 84:1–84:17



Leibniz International Proceedings in Informatics

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

84:2 An Empirical Evaluation of k -Means Coresets

Here we are given n points A with (potential unit) weights $w : A \rightarrow \mathbb{R}_{\geq 0}$ in some metric space with distance function dist and aim to find a set of k centers C such that

$$\text{cost}_A(C) := \frac{1}{n} \sum_{p \in A} \min_{c \in C} w(p) \cdot \text{dist}^z(p, c)$$

is minimized. The most popular variant of this problem is probably the k -means problem in d -dimensional Euclidean space where $z = 2$ and $\text{dist}(x, y) = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$.

A (k, ε) -coreset is now a subset $\Omega \subset A$ with weights $w : \Omega \rightarrow \mathbb{R}_{\geq 0}$ such that for any set of k centers C

$$\sup_C \max \left(\frac{\text{cost}_A(C)}{\text{cost}_\Omega(C)}, \frac{\text{cost}_\Omega(C)}{\text{cost}_A(C)} \right) \leq 1 + \varepsilon. \quad (1)$$

The coreset definition in Equation (1) provides an upper bound for the distortion of all candidate solutions i.e., all possible sets of k centers. A *weak coreset* is a relaxed guarantee that holds for optimal or nearly optimal clusterings of A instead of all clusterings.

In a long line of work spanning the last 20 years [4, 8, 10, 14, 15, 19, 20, 26, 25, 27, 29, 8, 33, 44], the size of coresets has been steadily improved with the current state of the art yielding a coreset with $\tilde{O}(k\varepsilon^{-2} \cdot \min(d, k, \varepsilon^{-2}))$ points for a distortion $D \leq (1 + \varepsilon)$ due to [13]¹.

While we have a good grasp of the theoretical guarantees of these algorithms, our understanding of the empirical performance is somewhat lacking. There exist a number of coreset implementations, but it is usually difficult to assess which implementation summarizes the data best. To accurately evaluate a given coreset, we would need to come up with a k clustering C which results in a maximal distortion. Solving this problem is likely difficult: related questions such as deciding whether a 3-dimensional point set A is an ε -net of a set B with respect to convex ranges is co-NP hard [24].

Due to this difficulty, a common heuristic for evaluating coresets is as follows [1, 22]. First, compute a coreset Ω with the available algorithm(s) using some input data A . Then, run an optimization algorithm on Ω to compute a k clustering. The *best* coreset algorithm is considered to be the one which yields a clustering with the smallest cost.

This practice has substantial drawbacks. The first is that this evaluation method conflates the two separate tasks of coreset construction and optimization. It is important to note that the first step of virtually all coreset algorithms is a low-cost (bicriteria) constant factor approximation, i.e. a solution with $\beta \cdot k$ clusters that costs at most $\alpha \cdot \text{OPT}$, where OPT is the cost of an optimal k clustering. Given that this initial solution has an α approximation to the cost, a routine calculation shows that the additive error of the coreset, i.e. the maximum difference $|\text{cost}_A(C) - \text{cost}_B(C)|$ over all solutions C is at most $O(\alpha) \cdot \text{cost}_A(C)$. In particular, in the case that the initial bicriteria approximation has $\alpha \ll 2$, which is not too difficult to achieve with more than k centers, any γ approximation algorithm will find solutions with approximation factor $O(\gamma + \alpha) \cdot \text{OPT}$. In particular, the distortion may be unbounded, for example if B only consists of the k centers, while simply returning B itself yields a low cost clustering. Thus, it is difficult to measure coreset quality in this way.

The second drawback is that this practice will mainly measure the performance of the optimization algorithm, rather than the performance of the coreset algorithms. During its execution it might simply not consider any solution with high distortion. For example, if the

¹ We use $\tilde{O}(x)$ to hide $\log^c x$ terms for any constant c .

approximation factor γ of the solution returned by the algorithm is large then this solution (as well as any even higher cost solution considered during the algorithm's execution) will have a low distortion.

The third drawback of this evaluation method is that it does not consider the main use cases of coresets, nor the full power of their guarantee. Indeed, if speeding up the computation of an optimization algorithm, one would hardly need a strong coreset; approximating the cost of every candidate solution, as weaker coreset definitions (or indeed a bicriteria approximation) would be suitable as well. A coreset's main and most powerful feature is *composability*, i.e. given two disjoint point sets X and Y , the union of a coreset of X and a coreset of Y is a coreset. Composability is what enables coresets to scale to massively parallel computation models and enables simple streaming algorithms via the merge and reduce technique. To which degree a coreset is composable is generally not a property of an optimal clustering of the point set, as optimal solutions C_X of X or C_Y of Y may have little in common with an optimal solution of $X \cup Y$.

The purpose of this study is to systematically evaluate the quality of various coreset algorithms for k -means. As such, we develop a new evaluation procedure which estimates the distortion of coreset algorithms. On real-world data sets, we observe that while the evaluated coreset algorithms are generally able to find solutions with comparable costs, there is a stark difference in their distortions. This shows that differences between optimization and compression are readily observable in practice.

As a complement to our evaluation procedure on real-world data sets, we propose a benchmark framework for generating synthetic data sets. We argue why this benchmark has properties that results in hard instances for all known coreset constructions. We also show how to efficiently estimate the distortion of a candidate coreset on the benchmark.

2 Coreset Algorithms

Though the algorithms vary in details, coreset constructions come in one of the following two flavours:

1. **Movement-based constructions:** Such algorithms compute a coreset Ω with T points given some input point set A such that $\text{cost}_\Omega(C) \ll \text{OPT}$, where OPT is the cost of an optimal k -means clustering of A . The coreset guarantee then follows as a consequence of the triangle inequality. These algorithms all have an exponential dependency on the dimension d , and therefore have been overtaken by sampling-based methods. Nevertheless, these constructions are more robust to various constrained clustering formulations [28, 43] and continue to be popular. Examples from theory include [23, 26].
2. **Importance sampling:** Points are sampled proportionate to their impact on the cost of any given candidate solution. The idealized distribution samples proportionate to the sensitivity which for a point p is defined as $\text{sens}(p) := \sup_C \frac{\min_{c \in C} \text{dist}^2(p, c)}{\text{cost}_A(C)}$ and weighted by their inverse sampling probability. The sensitivities are hard to compute exactly but much work exists on how to find other distributions with very similar properties. In terms of theoretical performance, sensitivity sampling has largely replaced movement-based constructions, see for example [19, 33].

Of course, there exist algorithms that draw on techniques from both, see for example [15]. In what follows, we will survey implementations of various coreset constructions that we will evaluate later.

StreamKM++ [1]. The popular k -means++ algorithm [3] computes a set of centers K by iteratively sampling a point p in A proportionate to $\min_{q \in K} \text{dist}^2(p, q)$ and adding it to K . The procedure terminates once the desired number of centers has been reached. The

first center is typically picked uniformly at random. The StreamKM++ paper runs the k -means++ algorithms for T iterations, where T is the desired coreset size. At the end, every point q in K is weighted by the number of points in A closest to it. While the construction has elements of importance sampling, the analysis is largely movement-based. The provable bound required for the algorithm to compute a coreset is $O\left(\frac{k \log n}{\delta^{d/2} \epsilon^d} \cdot \log^{d/2} \frac{k \log n}{\delta^{d/2} \epsilon^d}\right)$. Despite its simplicity, its running time compares unfavourably to all other constructions.

BICO [22]. BICO combines the very fast, but poor quality clustering algorithm BIRCH [47] with the movement-based analysis from [23, 26]. The clustering is organized by way of a hierarchical decomposition: When adding a point p to one of the coreset points Ω at level i , it first finds the closest point q in Ω . If p is too far away from q , a new cluster is opened with center at p . Otherwise p is either added to the same cluster as q , or, if adding p to q 's cluster increases the clustering cost beyond a certain threshold, the algorithm attempts to add p to the child-clusters of q . The procedure then continues recursively. The provable bound required for the algorithm to compute a coreset is $O(k\epsilon^{-d-2} \log n)$.

Ray Maker [25]. The algorithm computes an initial solution with k centers which is a constant factor approximation of the optimal clustering. Around each center, $O(1/\epsilon^{d-1})$ random rays are created which span the hyperplane. Next, each point $p \in A$ is snapped to its closest ray resulting in a set of one-dimensional points associated with each ray. Afterwards, a coreset is created for each ray by computing an optimal 1D clustering with k^2/ϵ^2 centers and weighing each center by the number of points in each cluster. The final coreset is composed of the coresets computed for all the rays. The provable bound required for the algorithm to compute a coreset is $O(k^3 \cdot \epsilon^{-d-1})$. The algorithm has recently received some attention due to its applicability to the fair clustering problem [28].

Sensitivity Sampling [19]. The simplest implementation of sensitivity sampling first computes an $(O(1), O(1))$ bicriteria approximation², for example by running k -means++ for $2k$ iterations [46]. Let K be the $2k$ clustering thus computed and let K_i be an arbitrary cluster of K with center q_i . Subsequently, the algorithm picks points proportionate to $\frac{\text{dist}^2(p, q)}{\text{cost}_{K_i}(\{q_i\})} + \frac{1}{|K_i|}$ and weighs any point by its inverse sampling probability. Let $|\hat{K}_i|$ be the estimated number of points in the sample. Finally, the algorithm weighs each q_i by $(1 + \epsilon) \cdot |K_i| - |\hat{K}_i|$. The provable bound required for the algorithm to compute a coreset is $\tilde{O}(kd\epsilon^{-4})$ ([19]), $\tilde{O}(k\epsilon^{-6})$ ([29]), or $\tilde{O}(k^2\epsilon^{-4})$ ([8]).

Group Sampling [15]. First, the algorithm computes an $O(1)$ approximation (or a bicriteria approximation) K . Subsequently, the algorithm preprocesses the input into groups such that (1) for any two points $p, p' \in K_i$, their cost is identical up to constant factors and (2) for any two clusters K_i, K_j , their cost is identical up to constant factors. In every group, Group Sampling now samples points proportionate to their cost. The authors of [15] show that there always exist a partitioning into $\log^2 1/\epsilon$ groups. Points not contained in a group are snapped to their closest center q in K . q is weighted by the number of points snapped to it. The provable bound required for the algorithm to compute a coreset is $\tilde{O}(k\epsilon^{-2} \min(d, k, \epsilon^{-2}))$ ([13]). While this improves over sensitivity sampling, it is generally slower and not as easy to implement.

² An (α, β) bicriteria approximation computes an α approximation using $\beta \cdot k$ many centers.

Finally, we note that some of the more popular algorithms in theory have not been mentioned here. For example, Chen’s [10] construction is particularly popular among theoreticians. The Group Sampling algorithm by [15] is an extension and improvement of Chen’s method. Thus, the performance of Group Sampling is also indicative of Chen’s algorithm.

Dimension Reduction

Finally, we also combine coresets with a variety of dimension reduction techniques. Starting with [17], a series of results [4, 5, 6, 7, 12, 16, 20, 21, 32, 37, 44] explored the possibility of using dimension reduction methods for k -clustering, with a particular focus on principal component analysis (PCA) and random projections. The seminal paper by Feldman, Schmidt, and Sohler [20] was the first to use dimension reduction to obtain smaller coresets for k -means. Movement-based coresets in particular often have an exponential dependency on the dimension, which can be alleviated with some form of dimension reduction, both in theory [43] and in practice [31]. There are essentially two main dimension reduction techniques for coresets.

Principal Component Analysis. Feldman, Schmidt, and Sohler [20] showed that projecting an input A onto the first $O(k/\varepsilon^2)$ principal components is a coreset. This coreset still consists of n points, but they now lie in low dimension. The analysis was subsequently tightened by [12] and extended to other center-based cost functions by [44]. Although its target dimension is generally worse than those based on random projections and terminal embeddings, there is nevertheless reasons for using PCA regardless: It removes noise and thus may make it easier to compute a high quality coreset. For more applications of PCA to k -means clustering, we refer to

Terminal Embeddings. Given a set of points A in \mathbb{R}^D , a terminal embedding $f : \mathbb{R}^D \rightarrow \mathbb{R}^d$ preserves the pairwise distance between any point $p \in A$ and any point $q \in \mathbb{R}^D$ up to a $(1 \pm \varepsilon)$ factor. The statement is related to the famous Johnson-Lindenstrauss lemma but it is stronger as it does not apply to only the pairwise distances of A . Nevertheless, the same target dimension is sufficient. Terminal embeddings were studied by [11, 18, 35, 42], with Narayanan and Nelson [42] achieving an optimal target dimension of $O(\varepsilon^{-2} \log n)$, where n is the number of points. We note that terminal embeddings, combined with an iterative application of the coreset construction from [8], can reduce the target dimension to a factor $\tilde{O}(\varepsilon^{-2} \log k)$. This is mainly of theoretical interest, as in practice the deciding factor wrt the target dimension is the precision, rather than dependencies on $\log n$ and $\log k$. For applications to coresets, we refer to [4, 15, 29]. For an empirical evaluation of random projections, which form the basis of all known terminal embeddings, we refer to Venkatasubramanian and Wang [45].

3 Benchmark Construction

In this section, we describe our benchmark. We start by describing the aims of the benchmark, followed by giving the construction. Our aim is to generate a data set containing many clusterings with the following properties.

1. The benchmark has many clusterings that, in a well defined sense, are highly dissimilar. Specifically, we want the overlap between any two clusters of different clusterings to be small.

2. The different clusterings have very similar and low cost. This ensures that despite the solutions being different in terms of composition and center placement, a good coreset has to consider them equally regarding distortion.
3. The clusterings are induced by a minimal cost assignment of input points to a set of centers in \mathbb{R}^d . This final property ensures that the coreset guarantee has to apply to these clusterings.

To generate the benchmark, we now use the following construction. The benchmark has a parameter α which controls the number of points and dimensions of the generated data instance. For a given value of k , the benchmark instance consists of $n = k^\alpha$ points and $d = \alpha \cdot k$ dimensions, i.e. we will construct an $n \times d$ matrix A where every row corresponds to an input point and every column corresponds to one of the dimensions.

Let $\mathbf{1}_k$ be the k -dimensional all-one vector and v_i^1 be the k -dimensional vector with entries $(v_i^1)_j = \begin{cases} -\frac{1}{k} & \text{if } i \neq j \\ \frac{k-1}{k} & \text{if } i = j \end{cases}$. For $\ell \leq \alpha$, recursively define the k^ℓ dimensional vector

$$v_i^\ell = v_i^{\ell-1} \otimes \mathbf{1}_k, \text{ where } \otimes \text{ denotes the Kronecker product, i.e. } v_i^{\ell-1} \otimes \mathbf{1}_k = \begin{bmatrix} (v_i^{\ell-1})_1 \cdot \mathbf{1}_k \\ (v_i^{\ell-1})_2 \cdot \mathbf{1}_k \\ \vdots \\ (v_i^{\ell-1})_{k^{\ell-1}} \cdot \mathbf{1}_k \end{bmatrix}.$$

Finally, set the t -th column of A , for $t = a \cdot k + b$, $a \in \{0, \dots, \alpha - 1\}$ and $b \in \{1, \dots, k\}$, to be $\mathbf{1}_{k^{\alpha-a+1}} \otimes v_b^{a+1}$.

To get a better feel for the construction, we have given two small example instances for $k = 2$ and $k = 3$ in Figure Figure 1.

$$\begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \quad \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

■ **Figure 1** Benchmark construction for $k = 2$ and $\alpha = 3$ (left) and $k = 3$ and $\alpha = 2$ (right).

Properties of the Benchmark

We now summarize the key properties of the benchmark. To this end, we require a few notions. Let A be the input matrix. We slightly abuse notation and refer to A_i as both the i th point as well as the i th row of the matrix A . For a clustering $\mathcal{C} = \{C_1, \dots, C_k\}$, we define that the $n \times k$ indicator matrix \tilde{X} induced by \mathcal{C} via $\tilde{X}_{i,j} = \begin{cases} 1 & \text{if } A_i \in C_j \\ 0 & \text{else.} \end{cases}$ Furthermore, we

will also use the $n \times k$ normalized clustering matrix X defined as $X_{i,j} = \begin{cases} \frac{1}{\sqrt{|C_j|}} & \text{if } A_i \in C_j \\ 0 & \text{else.} \end{cases}$

We also recall the following lemma which will allow us to express the k -means cost of a clustering \mathcal{C} with optimally chosen centers in terms of the cost of X and A .

► **Lemma 1** (Folklore). *Let A be an arbitrary set of points and let $\mu(A) = \frac{1}{|A|} \sum_{p \in A} p$ be the mean. Then $\sum_{p \in A} \|p - c\|^2 = |A| \cdot \|\mu(A) - c\|^2 + \sum_{p \in A} \|p - \mu(A)\|^2$ for any point c .*

This lemma proves that for any given cluster C_j , the mean is the optimal choice of center. We also note that any two distinct columns of X are orthogonal. Furthermore $\frac{1}{n} \mathbf{1}\mathbf{1}^T A$ copies the mean into every entry of A . Combining these two observations, we see that the matrix $XX^T A$ maps the i th row of A onto the mean of the cluster it is assigned to. Finally, define the Frobenius norm of an $n \times d$ A by $\|A\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^d A_{i,j}^2}$. Then the k -means cost of the clustering \mathcal{C} is precisely $\|A - XX^T A\|_F^2$.

We also require the following distance measure on clusterings as proposed by Meila [38, 39]. Given two clusterings \mathcal{C} and \mathcal{C}' , the $k \times k$ confusion matrix M is defined as $M_{i,j} = |C_i \cap C'_j|$. Furthermore for the indicator matrices \tilde{X} and \tilde{X}' induced by \mathcal{C} and \mathcal{C}' we have the identity $M = \tilde{X}^T \tilde{X}'$. Denote by Π_k the set of all permutations over k elements. Then the distance between \mathcal{C} and \mathcal{C}' is defined as $d(\mathcal{C}, \mathcal{C}') = 1 - \frac{1}{n} \max_{\pi \in \Pi_k} \sum_{i=1}^k M_{i,\pi(i)}$. Observe that for clusters that are identical, their distance is 0. The maximum distance between any two k clusterings is always $\frac{k-1}{k}$.

The solutions we consider are given as follows. For the columns $a \cdot k + 1, \dots, (a+1) \cdot k$, we define the clustering $\mathcal{C}^a = \{C_1^a, \dots, C_k^a\}$ with $A_i \in C_j^a$ if and only if $A_{i,j} > 0$. Let \tilde{X}^a and X^a denote the indicator matrix and clustering matrix, respectively, as induced by \mathcal{C}^a . These clusterings satisfy the properties we stated at the beginning of this section, that is:

1. The distance between these clustering is $1 - \frac{1}{k}$, i.e. it is maximized.
2. The clusterings have equal cost and the centers in each clustering have equal cost.
3. The clusterings are induced by a set of centers in \mathbb{R}^d .

Benchmark Evaluation

We now describe how we use the benchmark to measure the distortion of a coresets. Assume for now that the coresets are subsets of the original input points. The extension to coresets that do not consist of input points is described at the end of this section.

Consider the clustering $\mathcal{C}^a = \{C_1^a, \dots, C_k^a\}$ for some a and let Ω with weights $w : \Omega \rightarrow \mathbb{R}_{\geq 0}$ be the coresets and let $\delta > 0$ be a parameter. Note that there are α many such clusterings, for each value of a . We use $w(C_i^a \cap \Omega) := \sum_{p \in C_i^a \cap \Omega} w(p)$ to denote the mass of points of C_i^a in Ω . For every cluster C_i^a with $w(C_i^a \cap \Omega) \geq |C_i^a|(1 - \delta)$, we place a center at $\mu(C_i^a)$. Conversely, if $w(C_i^a \cap \Omega) < |C_i^a|(1 - \delta)$, we do not place a center at $\mu(C_i^a)$. We call such clusters *deficient*. Let \mathcal{S} be the centers of these deficient clusters.

We now compare the cost as computed on the coresets and the true cost of \mathcal{S} . Due to Lemma 1 and the fact that all clusters have equal cost, we may write for any deficient cluster C_i^a $\text{cost}_{C_i^a}(\mathcal{S}) = \text{cost}_{C_j^a}(\{\mu(C_j^a)\}) + k^{\alpha-1} \|\mu(C_j^a) - \mu(C_i^a)\|_2^2$, where C_h^a is a non-deficient cluster. Thus, the cost is $\text{cost}_{C_i^a}(\mathcal{S}) \approx (1 + \frac{2}{\alpha}) \cdot \text{cost}_{C_j^a}(\{\mu(C_j^a)\})$.

Conversely, the cost on the coresets is

$$\text{cost}_{\Omega \cap C_i^a}(\mathcal{S}) \approx \frac{w(C_i^a \cap \Omega)}{\text{cost}_{C_j^a}(\{\mu(C_j^a)\})} \left(1 + \frac{2}{\alpha}\right) \cdot \text{cost}_{C_j^a}(\{\mu(C_j^a)\}).$$

Thus for each deficient clustering individually, the distortion will be close to $\frac{k^{\alpha-1}}{w(C_i^a \cap \Omega)} \geq \frac{1}{1-\delta}$. If there are many deficient clusters, then this will also be the overall distortion. For all possible (suitably discretized) thresholds for deficiency, i.e. all values of δ , we can now identify the clustering \mathcal{C}^a with a maximum number of deficient clusters and use the aforementioned construction to get a lower bound on the distortion.

To extend this evaluation to coresets where the points are not part of the input, we consider a point $p \in \Omega$ to be in C_i^a if it is closer to $\mu(C_i^a)$ than to $\mu(C_j^a)$.

4 Experiments

In this section, we present how we evaluated different algorithms. First, we propose our evaluation procedure which gauges the quality of coresets. Then, we describe the data sets used for the empirical evaluation and our experimental setup. Finally, we detail the outcome of the experiments and our interpretation of the results.

Evaluation Procedure

Accurately evaluating a k -means coreset of a real-world data set requires constructing a solution (a set of k centers) which results in a maximal distortion. Finding such a solution, however, is difficult. Instead, we can estimate the quality of a given coreset by finding meaningful candidate solutions.

A first attempt can be to randomly generate candidate solutions. It is not readily apparent how to define a distribution of meaningful solutions from which to sample. One could, for instance, generate k random points inside the convex hull or the minimum enclosing ball (MEB) of a coreset Ω . Convex hulls in high dimensions are infeasible to compute, so we sample a center by choosing random convex combination of the centers of the initial bicriteria approximation computed for every coreset. A better way to generate candidate solutions turns out to be k -means++, where we sample k points with respect to the k -means++ distribution and use the resulting centers as a solution. The main advantage of this approach is that k -means++ can uncover natural cluster structures in the data, which uniform sampling generally does not. For all variants, we generated 5 candidate solutions, where the candidate solution with the largest distortion being a lower bound for the true distortion of the coreset.

Given the usefulness of evaluating coresets on real-world data sets, it can be tricky to gauge the general performance of coreset algorithms using only a small selection of data sets. For this reason, we used our benchmark to complement the evaluation on real-world data sets. The benchmark accomplishes two important tasks. First, the benchmark allows us to quickly find a bad solution because both good and bad clusterings are known a priori. It is unclear how to find bad clusterings for real-world data sets. Second, it is easier to make a fair comparison of different coreset constructions because the benchmark is known to generate hard instances for all known coreset algorithms. This cannot be said for real-world data sets. For the benchmark, we computed the distortion following the evaluation procedure described in Section 3.

Every randomized coreset construction was repeated 10 times. We aggregated the reported maximum distortions for every run by taking the average over all 10 evaluations. It is important to not aggregate the distortions here by taking the maximum over all runs: If one run of the coreset algorithm fails but the others succeed, then such an aggregation predicts far worse distortion than what we could typically expect.

Data sets

We conducted experiments on five real-world data sets *Census*, *Coverttype*, *Tower*, *Caltech*, *NYTimes*, and four instances of our benchmark. Benchmark instances were generated to match approximately the sizes of the real-world data sets. The sizes of the considered data sets are given in Table 1.

■ **Table 1** The sizes of the real-world datasets used for the experimental evaluation.

	Data points	Dimensions
<i>Caltech</i>	3,680,458	128
<i>Census</i>	2,458,285	68
<i>Covertypes</i>	581,012	54
<i>NYTimes</i>	500,000	102,660
<i>Tower</i>	4,915,200	3

■ **Table 2** The parameter values and the sizes of the benchmark instances used for the experimental evaluation.

k	α	Data points	Dimensions
10	6	1,000,000	60
20	5	3,200,000	100
30	4	810,000	120
40	4	2,560,000	160

The *Census*³ dataset is a small subset of the Public Use Microdata Samples from 1990 US census. It consists of demographic information encoded as 68 categorical attributes of 2,458,285 individuals.

*Covertypes*⁴ is comprised of cartographic descriptions and forest cover type of four wilderness areas in the Roosevelt National Forest of Northern Colorado in the US. It consists of 581,012 records, 54 cartographic variables and one class variable. Although *Covertypes* was originally made for classification tasks, it is often used for clustering tasks by removing the class variable [1].

The data set with the fewest number of dimensions is *Tower*⁵. This data set consists of 4,915,200 rows and 3 features as it is a 2,560 by 1,920 picture of a tower on a hill where each pixel is represented by a RGB color value.

Inspired by [22], *Caltech* was created by computing SIFT features from the images in the Caltech101⁶ image database. This database contains pictures of objects partitioned into 101 categories. Disregarding the categories, we concatenated the 128-dimensional SIFT vectors from each image into one large data matrix with 3,680,458 rows and 128 columns.

*NYTimes*⁷ is a dataset composed of the bag-of-words (BOW) representations of 300,000 news articles from The New York Times. The vocabulary size of the text collection is 102,660. Due to the BOW encoding, *NYTimes* has a very large number of dimensions and is highly sparse. To make processing feasible, we reduced the number of dimensions to 100 using terminal embeddings.

³ [https://archive.ics.uci.edu/ml/datasets/US+Census+Data+\(1990\)](https://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990))

⁴ <https://archive.ics.uci.edu/ml/datasets/covertypes>

⁵ <http://homepages.uni-paderborn.de/frahling/coremeans.html>

⁶ http://www.vision.caltech.edu/Image_Datasets/Caltech101/

⁷ <https://archive.ics.uci.edu/ml/datasets/bag+of+words>

Preprocessing & Experimental Setup

To understand how denoising effects the quality of the outputted coresets, we applied Principal Component Analysis (PCA) on *Caltech*, *Census*, *Coverttype*, and *NYTimes* by using the k singular vectors corresponding to the largest singular values. We did not perform any preprocessing on *Tower* due to its low dimensionality.

We followed the same experimental procedure with respect to the choice of parameter values for the algorithms as prior works [1, 22]. For the target coreset size T , we experimented with $T = mk$ for $m = \{50, 100, 200, 500\}$. On *Caltech*, *Census*, *Coverttype*, and *NYTimes*, we used values k in $\{10, 20, 30, 40, 50\}$, while for *Tower* we used larger cluster sizes $k \in \{20, 40, 60, 80, 100\}$. On the benchmark, we used $k \in \{10, 20, 30, 40\}$.

We implemented Sensitivity Sampling, Group Sampling, Ray Maker, and StreamKM++ in C++. The source code can be found on GitHub⁸. For BICO, we used the authors' reference implementation⁹. The source code was compiled with gcc 9.3.0. The experiments were performed on a machine with Intel Core i9 10940X 3.3GHz 14-Core and 2x DDR4 PC3200 128GB RAM.

Outcome of Experiments

We observed that in the majority of our experiments, varying the coreset sizes does not significantly change the performance profiles of individual algorithms when comparing them against each other. Therefore, in the following sections, we focus on a cross-section of the experiments where $m = 200$ i.e., coreset sizes $T = 200k$. For numerical results including variances of all the experiments and tables containing distortions, costs and running times, we refer to the full version of this paper.

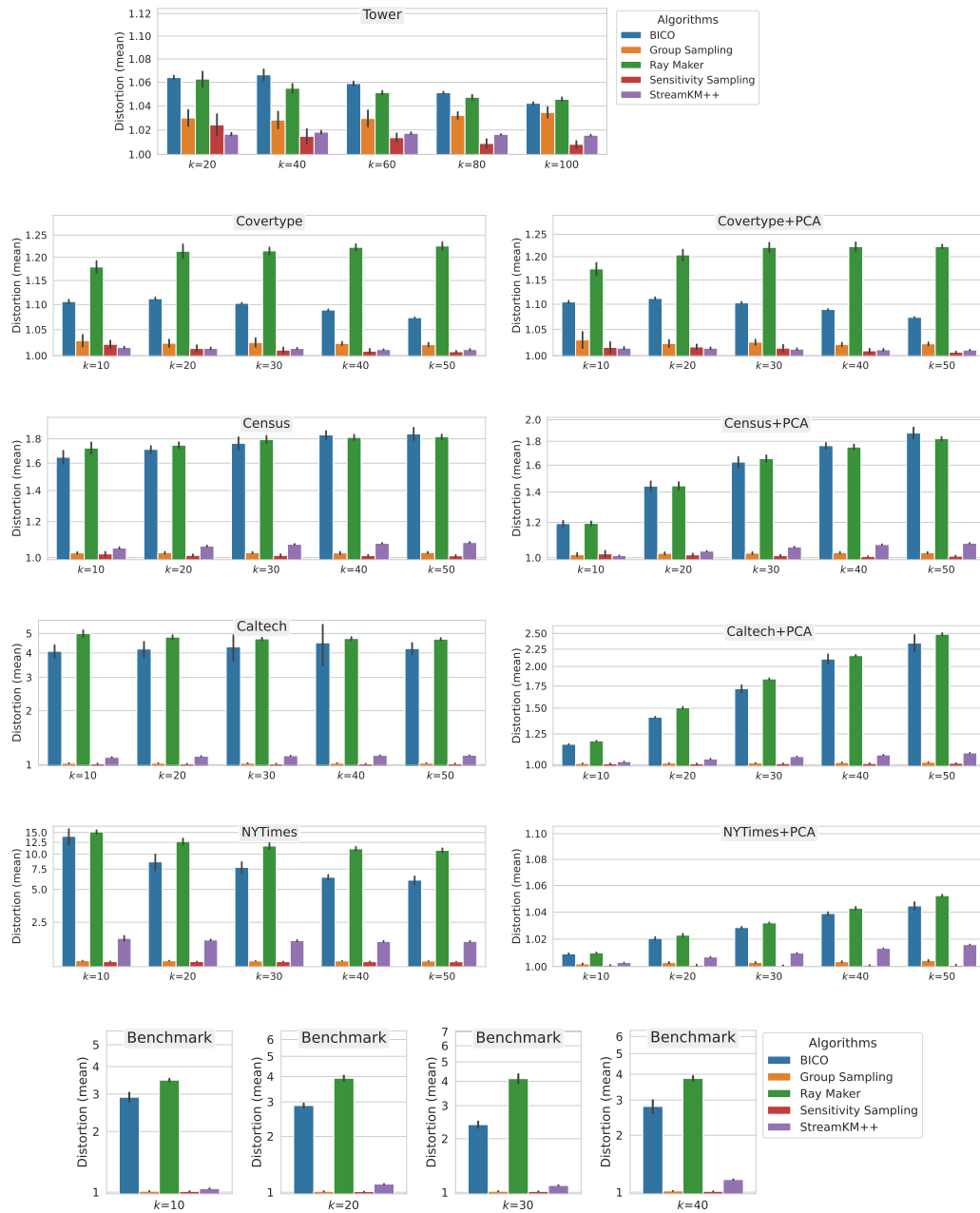
In Figure 2, we summarized the distortions of the experiments with coreset sizes $T = 200k$. All five algorithms are matched on the *Tower* dataset. The worst distortions across the algorithms are close to 1, and performance between the algorithms is negligible. The performance difference between sampling-based and movement-based methods become more pronounced as the number of dimensions increase. On *Coverttype* with its 54 features, Ray Maker performs the worst followed by BICO and Group Sampling while Sensitivity Sampling and StreamKM++ perform the best. Differences in performance are more noticeable on *Census*, *Caltech*, and *NYTimes* where methods based on importance sampling perform much better. Sensitivity Sampling and Group Sampling perform the best, StreamKM++ come in second while BICO and Ray Maker perform the worst across these data sets. On the *Benchmark*, Ray Maker is the worst while Sensitivity Sampling and Group Sampling are the best. StreamKM++ performs also very well compared to BICO.

Interpretation of Experimental Results

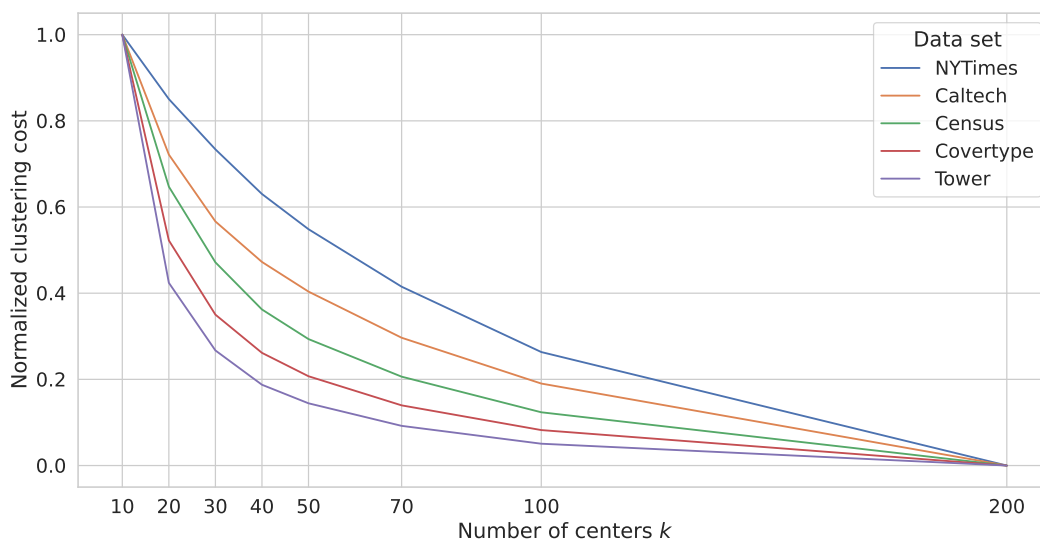
Optimization versus Compression. While all five algorithms are equally matched when optimizing on the candidate coresets, coreset quality performance differ significantly (see Figure 2). For all data sets, the obtained costs differed insignificantly for all values of k , irrespective of the coreset algorithm used, while distortions varied strongly, depending on the coreset algorithm.

⁸ <https://github.com/sheikhomar/eval-k-means-coresets>

⁹ <https://ls2-www.cs.tu-dortmund.de/grav/en/bico>



■ **Figure 2** The average distortions of the evaluated coreset algorithms with coreset size $T = 200k$ on five real-world data sets and on four benchmark instances. Black bars indicate standard deviations. Notice that the axis is non-linear as otherwise the bars for Sensitivity Sampling and Group Sampling would disappear on the plots as their distortions are close to 1.



■ **Figure 3** Depicts how clustering costs of five real-world data sets decrease as the number of centers increase. Plotting the cost curve allows us to study whether we can observe a difference between coreset construction and optimization in a data set when evaluating a coreset based on cost.

Nevertheless, the cost drop with increasing values of k is a predictor for the quality of certain coresets. It is not uncommon for the k -means cost of real-world data sets to drop significantly for larger values of k . Figure 3 illustrates this behavior for several real-world data sets. The more the curve bends, the less of a difference there is between computing a coreset and a clustering with low cost. For data sets with an L-shaped cost curve, a coreset algorithm adding more centers to the coreset will seem to be performing well when evaluating it based on the outcome of the optimization. *Tower* is a good example of a data set where optimization is very close to compression. Its cost curve bends the most which indicates that adding more centers help reduce the cost. One of the strengths of the benchmark is that there is no way of reducing the cost without capturing the right subclusters within a benchmark instance. This means that the cost does not decrease markedly beyond a certain value of k even if more centers are added.

For BICO, Ray Maker, and StreamKM++, there is a correlation between the steepness of the cost curve for a data set and the distortion of the generated coreset. On data sets where the curve is less steep, we observed higher distortions. The effect is more pronounced for BICO and Ray Maker than for StreamKM++. Importance sampling approaches (Group Sampling and Sensitivity Sampling) seem to be free from this behavior as they consistently generate high quality coresets irrespective of the shape of cost curve.

Movement-based versus Sampling-based Approaches. In general, movement-based constructions perform the worst in terms of coreset quality. We observed that BICO and Ray Maker have the highest distortions across all data sets including on the benchmark instances. Among the sampling-based algorithms, Sensitive Sampling performs well with Group Sampling generally being competitive. This runs contrary to theory where Group Sampling has the better (currently known) theoretical bounds. StreamKM++ is an interesting case. Like the movement-based methods, its distortion increases with the dimension. Nevertheless, it generally performs significantly better than BICO and Ray Maker. This can be attributed to the fact that the coreset produced by StreamKM++ consists entirely of

k -means++ centers weighted by the number of points of a minimal cost assignment. This is similar to movement-based algorithms such as BICO. Nevertheless, it also retains some of the performance from pure importance schemes.

In practice as well as in theory, the distortion of movement-based algorithms are affected by the dimension. By comparison, sampling-based algorithms are affected very little. Theoretically, there should not exist a difference, as the sampling bounds are independent of the dimension. What little effect can be observed is likely due to PCA making it easier to find low cost solutions that form the backbone of all coresets constructions. StreamKM++ is an interesting case, as it is still affected by the dimension, though less than the other movement based methods.

A notable exception is the benchmark. Here, sensitivity sampling generally found the lowest cost clustering, with BICO finding the second lowest cost clustering. This happens *despite* BICO generally having a worse distortion than for example Group Sampling or StreamKM++.

Impact of PCA. On almost all our data sets, the performance improves when input data is preprocessed with PCA, especially for the movement-based algorithms. Empirically, the more noise is removed (i.e., small k value), the lower the distortion. Notice that k is the number of principal components that the input data is projected on to. The rest of the low variance components are treated as noise and removed. Method utilizing sampling (Group Sampling, Sensitivity Sampling and StreamKM++) are less effected by the preprocessing step. On *Covertime*, PCA does not change the distortions by much because almost all the variance in the data is explained by the first five principal components. On *Caltech* and *NYTimes*, the quality of the coresets by BICO and Ray Maker improves greatly because the noise removal is more aggressive. Even if the quality is much better for movement-based coresets constructions due to PCA, importance sampling methods are still superior when it comes to the quality of the compression. Summarizing, all methods benefit from PCA, and in case of movement-based constructions, we consider PCA a necessary preprocessing step. For the sampling-based methods, the computational expense of using PCA in preprocessing does not seem justify the comparatively meager gains in coresets distortion.

5 Conclusion

In this work, we studied how to assess the quality of k -means coresets computed by state-of-the-art algorithms. Previous work generally measured the quality of optimization algorithms run on the coresets, which we empirically observed to be a poor indicator of coresets quality. For real-world data sets, we sampled candidate clusterings and evaluated the worst case distortion on them. Complementing this, we also proposed a benchmark framework which generates hard instances for known k -means coresets algorithms. Our experiments indicate a general advantage for algorithms based on importance sampling over movement-based methods. Despite movement-based methods running on very efficient code, it is necessary to complement them with rather expensive dimension reduction methods, rendering what efficiency they might have over importance sampling somewhat moot.

Two results bear further investigation. First, the currently known provable coresets sizes for Sensitivity Sampling are worse than those provable via Group Sampling. Empirically, we observed the opposite: While Group Sampling is competitive, Sensitivity Sampling always outperforms it. Since Group Sampling requires somewhat cumbersome computational overhead, practical applications should prefer Sensitivity Sampling. In light of these results, a theoretical analysis for Sensitivity Sampling matching the performance of Group Sampling would be welcome.

The second point of interest focuses on the performance of StreamKM++. The distortion of this algorithm is significantly better than what one would expect from its theoretical analysis. Empirically, StreamKM++ is notably better than the other movement-based constructions across all data sets, and especially on high dimensional data. While it is not competitive to the pure importance sampling algorithms, there are several reasons for investigating it further. It essentially only requires running k -means++ for additional iterations, which is already a nearly ubiquitous algorithm for the k -means problem. Although the other sampling-based coreset algorithms can also be readily implemented, doing so might be cumbersome. In particular, the theoretically (but not empirically) best algorithm Group Sampling requires extensive preprocessing steps. This begs the question whether there exist a better theoretical analysis for StreamKM++.

In addition, StreamKM++ currently weighs each point by the number of points assigned to it. It may also be possible to improve the performance of the algorithm in both theory and practice by using a different weighting scheme. We leave this as an open problem for future research.

References

- 1 Marcel R. Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammensen, and Christian Sohler. Streamkm++: A clustering algorithm for data streams. *ACM Journal of Experimental Algorithmics*, 17(1), 2012. doi:10.1145/2133803.2184450.
- 2 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. In *Combinatorial and computational geometry, MSRI*, pages 1–30. University Press, 2005.
- 3 David Arthur and Sergei Vassilvitskii. k -means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 1027–1035, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283494>.
- 4 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, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1039–1050, 2019. doi:10.1145/3313276.3316318.
- 5 Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. Unsupervised feature selection for the k -means clustering problem. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 153–161, 2009. URL: <http://papers.nips.cc/paper/3724-unsupervised-feature-selection-for-the-k-means-clustering-problem>.
- 6 Christos Boutsidis, Anastasios Zouzias, and Petros Drineas. Random projections for k -means clustering. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada.*, pages 298–306, 2010. URL: <http://papers.nips.cc/paper/3901-random-projections-for-k-means-clustering>.
- 7 Christos Boutsidis, Anastasios Zouzias, Michael W. Mahoney, and Petros Drineas. Randomized dimensionality reduction for k -means clustering. *IEEE Trans. Information Theory*, 61(2):1045–1062, 2015. doi:10.1109/TIT.2014.2375327.
- 8 Vladimir Braverman, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in excluded-minor graphs and beyond. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2679–2696. SIAM, 2021. doi:10.1137/1.9781611976465.159.

- 9 Timothy M. Chan. Dynamic coresets. *Discret. Comput. Geom.*, 42(3):469–488, 2009. doi:10.1007/s00454-009-9165-3.
- 10 Ke Chen. On coresets for k-median and k-means clustering in metric and Euclidean spaces and their applications. *SIAM J. Comput.*, 39(3):923–947, 2009.
- 11 Yeshwanth Cherapanamjeri and Jelani Nelson. Terminal embeddings in sublinear time. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1209–1216. IEEE, 2021. doi:10.1109/FOCS52979.2021.00118.
- 12 Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 163–172, 2015.
- 13 Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, and Chris Schwiegelshohn. Towards optimal lower bounds for k-median and k-means coresets. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1038–1051. ACM, 2022. doi:10.1145/3519935.3519946.
- 14 Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. Improved coresets and sublinear algorithms for power means in euclidean spaces. In Marc’Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 21085–21098, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/b035d6563a2adac9f822940c145263ce-Abstract.html>.
- 15 Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 169–182. ACM, 2021.
- 16 Vincent Cohen-Addad and Chris Schwiegelshohn. On the local structure of stable clustering instances. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 49–60, 2017. doi:10.1109/FOCS.2017.14.
- 17 Petros Drineas, Alan M. Frieze, Ravi Kannan, Santosh Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Machine Learning*, 56(1-3):9–33, 2004. doi:10.1023/B:MACH.0000033113.59016.96.
- 18 Michael Elkin, Arnold Filtser, and Ofer Neiman. Terminal embeddings. *Theor. Comput. Sci.*, 697:1–36, 2017. doi:10.1016/j.tcs.2017.06.021.
- 19 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 569–578, 2011.
- 20 Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca, and projective clustering. *SIAM J. Comput.*, 49(3):601–657, 2020. doi:10.1137/18M1209854.
- 21 Zhili Feng, Praneeth Kacham, and David P. Woodruff. Dimensionality reduction for the sum-of-distances metric. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 3220–3229. PMLR, 2021. URL: <http://proceedings.mlr.press/v139/feng21a.html>.
- 22 Hendrik Fichtenberger, Marc Gillé, Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. BICO: BIRCH meets coresets for k-means clustering. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 481–492, 2013.


- 23 Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–217, 2005.
- 24 Panos Giannopoulos, Christian Knauer, Magnus Wahlström, and Daniel Werner. Hardness of discrepancy computation and ϵ -net verification in high dimension. *J. Complex.*, 28(2):162–176, 2012. doi:10.1016/j.jco.2011.09.001.
- 25 Sariel Har-Peled and Akash Kushal. Smaller coresets for k -median and k -means clustering. *Discret. Comput. Geom.*, 37(1):3–19, 2007. doi:10.1007/s00454-006-1271-x.
- 26 Sariel Har-Peled and Soham Mazumdar. On coresets for k -means and k -median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 291–300, 2004.
- 27 Lingxiao Huang, Shaofeng H.-C. 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. doi:10.1109/FOCS.2018.00082.
- 28 Lingxiao Huang, Shaofeng H.-C. Jiang, and Nisheeth K. Vishnoi. Coresets for clustering with fairness constraints. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7587–7598, 2019. URL: <https://proceedings.neurips.cc/paper/2019/hash/810dfbbebb17302018ae903e9cb7a483-Abstract.html>.
- 29 Lingxiao Huang and Nisheeth K. Vishnoi. Coresets for clustering in euclidean spaces: importance sampling is nearly optimal. 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 1416–1429. ACM, 2020. doi:10.1145/3357713.3384296.
- 30 Piotr Indyk, Sepideh Mahabadi, Shayan Oveis Gharan, and Alireza Rezaei. Composable core-sets for determinant maximization problems via spectral spanners. 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 1675–1694. SIAM, 2020. doi:10.1137/1.9781611975994.103.
- 31 Jan-Philipp W. Kappmeier, Daniel R. Schmidt, and Melanie Schmidt. Solving k -means on high-dimensional big data. In *Experimental Algorithms - 14th International Symposium, SEA 2015, Paris, France, June 29 - July 1, 2015, Proceedings*, pages 259–270, 2015. doi:10.1007/978-3-319-20086-6_20.
- 32 Amit Kumar and Ravindran Kannan. Clustering with spectral norm and the k -means algorithm. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 299–308, 2010. doi:10.1109/FOCS.2010.35.
- 33 Michael Langberg and Leonard J. Schulman. Universal ϵ -approximators for integrals. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 598–607, 2010. doi:10.1137/1.9781611973075.50.
- 34 Alaa Maaouf, Ibrahim Jubran, and Dan Feldman. Fast and accurate least-mean-squares solvers. In *Advances in Neural Information Processing Systems*, pages 8307–8318, 2019.
- 35 Sepideh Mahabadi, Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Nonlinear dimension reduction via outer bi-lipschitz extensions. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1088–1101, 2018. doi:10.1145/3188745.3188828.

- 36 Tung Mai, Cameron Musco, and Anup Rao. Coresets for classification – Simplified and strengthened. In Marc’Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 11643–11654, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/6098ed616e715171f0dabad60a8e5197-Abstract.html>.
- 37 Konstantin Makarychev, Yury Makarychev, and Ilya P. 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 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1027–1038, 2019. doi:10.1145/3313276.3316350.
- 38 Marina Meila. Comparing clusterings: an axiomatic view. In Luc De Raedt and Stefan Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 577–584. ACM, 2005. doi:10.1145/1102351.1102424.
- 39 Marina Meila. The uniqueness of a good optimum for k -means. In William W. Cohen and Andrew W. Moore, editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 625–632. ACM, 2006. doi:10.1145/1143844.1143923.
- 40 Alexander Munteanu and Chris Schwiegelshohn. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *Künstliche Intell.*, 32(1):37–53, 2018. doi:10.1007/s13218-017-0519-3.
- 41 Alexander Munteanu, Chris Schwiegelshohn, Christian Sohler, and David P. Woodruff. On coresets for logistic regression. 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, December 3-8, 2018, Montréal, Canada*, pages 6562–6571, 2018. URL: <https://proceedings.neurips.cc/paper/2018/hash/63bfd6e8f26d1d3537f4c5038264ef36-Abstract.html>.
- 42 Shyam Narayanan and Jelani Nelson. Optimal terminal dimensionality reduction in euclidean space. 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 1064–1069. ACM, 2019. doi:10.1145/3313276.3316307.
- 43 Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k -means. In *Approximation and Online Algorithms - 17th International Workshop, WAOA 2019, Munich, Germany, September 12-13, 2019, Revised Selected Papers*, pages 232–251, 2019. doi:10.1007/978-3-030-39479-0_16.
- 44 Christian Sohler and David P. Woodruff. Strong coresets for k -median and subspace approximation: Goodbye dimension. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 802–813, 2018. doi:10.1109/FOCS.2018.00081.
- 45 Suresh Venkatasubramanian and Qiushi Wang. The johnson-lindenstrauss transform: An empirical study. In Matthias Müller-Hannemann and Renato Fonseca F. Werneck, editors, *Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2011, Holiday Inn San Francisco Golden Gateway, San Francisco, California, USA, January 22, 2011*, pages 164–173. SIAM, 2011.
- 46 Dennis Wei. A constant-factor bi-criteria approximation guarantee for k -means++. 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 604–612, 2016.
- 47 Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: A new data clustering algorithm and its applications. *Data Min. Knowl. Discov.*, 1(2):141–182, 1997. doi:10.1023/A:1009783824328.

An Improved Algorithm for Finding the Shortest Synchronizing Words

Marek Szykuła  

Faculty of Mathematics and Computer Science, University of Wrocław, Poland

Adam Zyzik 

Faculty of Mathematics and Computer Science, University of Wrocław, Poland

Abstract

A synchronizing word of a deterministic finite complete automaton is a word whose action maps every state to a single one. Finding a shortest or a short synchronizing word is a central computational problem in the theory of synchronizing automata and is applied in other areas such as model-based testing and the theory of codes. Because the problem of finding a shortest synchronizing word is computationally hard, among *exact* algorithms only exponential ones are known. We redesign the previously fastest known exact algorithm based on the bidirectional breadth-first search and improve it with respect to time and space in a practical sense. We develop new algorithmic enhancements and adapt the algorithm to multithreaded and GPU computing. Our experiments show that the new algorithm is multiple times faster than the previously fastest one and its advantage quickly grows with the hardness of the problem instance. Given a modest time limit, we compute the lengths of the shortest synchronizing words for random binary automata up to 570 states, significantly beating the previous record. We refine the experimental estimation of the average reset threshold of these automata. Finally, we develop a general computational package devoted to the problem, where an efficient and practical implementation of our algorithm is included, together with several well-known heuristics.

2012 ACM Subject Classification Theory of computation → Algorithm design techniques; Theory of computation → Formal languages and automata theory

Keywords and phrases Černý conjecture, reset threshold, reset word, subset checking, synchronizing automaton, synchronizing word

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.85

Related Version *Full Version*: <https://arxiv.org/abs/2207.05495>

Supplementary Material *Software (Source Code)*: <https://github.com/marekesz/synchrowords>
archived at `swh:1:dir:2a7194d536e2252dc2a6e18ec5505eb86cb44481`

Funding This work was supported in part by the National Science Centre, Poland under project number 2021/41/B/ST6/03691. The experiments were run on a computational grid of the Institute of Computer Science, University of Wrocław, funded by National Science Centre, Poland, under project number 2019/35/B/ST6/04379.

1 Introduction

A *deterministic finite complete semi-automaton* (called simply an *automaton*) is a 3-tuple (Q, Σ, δ) , where Q is a finite set of *states*, Σ is an *input alphabet*, and $\delta: Q \times \Sigma \rightarrow Q$ is a completely defined *transition function*. The transition function is naturally extended to a function $Q \times \Sigma^* \rightarrow Q$. Throughout the paper, by n we denote the number of states in Q and by k we denote the size of the input alphabet Σ . A word is *reset* (or *synchronizing*) if $|\delta(Q, w)| = 1$; in other words, for every two states $p, q \in Q$ we have $\delta(q, w) = \delta(p, w)$. An automaton that admits a reset word is called *synchronizing*.



© Marek Szykuła and Adam Zyzik;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 85; pp. 85:1–85:15
Leibniz International Proceedings in Informatics



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

The classical synchronization problem is, for a given synchronizing automaton, to find a reset word. Preferably, this word should be as short as possible. Therefore, the main property of a synchronizing automaton is its *reset threshold*, which is the length of the shortest reset words. We denote the reset threshold by r . Synchronizing automata and the synchronization problem are known for both their theoretical properties and practical applications.

1.1 Theoretical Developments

On the theoretical side, there is a famous long-standing open problem from 1969 called the Černý conjecture; see an old [44] and a recent survey [18]. The conjecture claims that the reset threshold is at most $(n-1)^2$. If true, the bound would be tight, as the Černý automata meet the bound for each n [8].

Until 2017, the best known upper bound on the reset threshold was $(n^3 - n)/6 - 1 \sim 0.1666\dots n^3 + \mathcal{O}(n^2)$ ($n \geq 4$) [26] by the well-known Frankl-Pin's bound. The current best known upper bound is $\sim 0.1654n^3 + o(n^3)$ by Shitov [37], which was obtained by refining the previous improvement $\sim 0.1664n^3 + \mathcal{O}(n^2)$ by Szykuła [39]. Apart from that, better bounds were obtained for many special subclasses of automata. Synchronizing automata are also applied in other theoretical areas, e.g., matrix theory [14], theory of codes [7], Markov processes [43]. Several new results around the topic appear every year. Recently, a special journal issue was dedicated to the problem [45] for the occasion of the 50th anniversary of the problem.

Reset thresholds were also studied for the average case. Berlinkov showed that a random binary automaton is synchronizing with high probability [5]. Moreover, Nicaud showed that such an automaton with high probability has a reset threshold in $\mathcal{O}(n \log^3 n)$ [24]. Based on that, the upper bound $\mathcal{O}(n^{3/2+o(1)})$ on the expected reset threshold of a random binary automaton was obtained [4]. These studies were accompanied by experiments, and the best estimation obtained so far was $2.5\sqrt{n-5}$ [19].

There were also performed massive experiments directly aimed at verifying the Černý conjecture and other theoretical properties for automata with small numbers of states [9, 21, 42]. For all such studies, finding (the length of) a shortest reset word is a crucial problem.

1.2 Synchronization in Applications

Apart from the theory, the synchronization problem finds applications in practical areas, e.g., testing of reactive systems [29, 34], networks [17], robotics [1], and codes [15].

Automata are frequently used to model the behavior of systems, devices, circuits, etc. The idea of synchronization is natural: we aim to restore control over a device whose current state is not known or we do not want our actions to be dependent on it. For instance, for digital circuits, where we need to test the conformance of the system according to its model, each test is an input word and before we run the next one, we need to restart the device. In another setting, we are an observer who knows the structure of the automaton but does not see its current state and wants to eventually learn it by observing the input; once a reset word appears, the state is revealed unambiguously. See a survey [34] explaining synchronizing sequences and their generalization to automata with output: *homing sequences*, which allow determining the (hidden) current state of the automaton by additionally observing the generated automaton's output.

Another particular application comes from the theory of codes, where finite automata act as *decoders* of a compressed input. Synchronizing words can make a code resistant to errors, since if an error occurs, after reading such a word, decoding is restored to the correct path. See a book for the role of synchronization in the theory of codes [7] and recent works [6] explaining synchronization applied to prefix codes.

1.3 Algorithms Finding Reset Words

Determining the reset threshold is computationally hard. The decision problem, whether the reset threshold is smaller than a given integer, is NP-complete [10], and it remains hard even for very restrictive classes such as binary Eulerian automata [47]. The functional problems of computing the reset threshold and a shortest reset word are respectively $\text{FP}^{\text{NP}^{\lceil \log \rceil}}$ -complete and FP^{NP} -complete [25]. Moreover, approximating the reset threshold is hard even for approximation factors in $\mathcal{O}(n^{1-\varepsilon})$, for every $\varepsilon > 0$ [12]. This inapproximability also holds for subclasses related to prefix codes [33]. On the other hand, there exists a simple general $\mathcal{O}(n)$ -approximating polynomial algorithm [13]. It is open whether there exists a polynomial algorithm approximating within some sublinear factor, e.g., in $\mathcal{O}(n/\log n)$.

From the perspective of fixed-parameter tractability, the main parameter determining the hardness is the reset threshold itself (and the alphabet size, if not fixed). It plays a similar role as the number of variables in the SAT problem, yet, in contrast, it is not given but is the result to be computed. It is trivial to compute the reset threshold in time $\mathcal{O}(n \cdot k^r)$ simply by checking all words of length $1, 2, \dots, r$ (we need $\mathcal{O}(n)$ time for computing the image of a subset of Q under the action of one letter). This essentially cannot be much better, as assuming the Strong Exponential Time Hypothesis (SETH), the problem cannot be solved in time $\mathcal{O}^*((k - \varepsilon)^r)$, for every $\varepsilon > 0$ [11, Theorem 8], where \mathcal{O}^* suppresses all polynomial factors in the size of the input.

Therefore, exponential exact algorithms that hopefully find a shortest reset word faster in typical or average cases are used. Alternatively, there are many polynomial heuristics proposed that find a relatively short reset word in practice.

1.3.1 Exact Algorithms

In general, exact algorithms can be used for automata that are not too large. They also play an important role in testing heuristics, providing the baseline for comparison (e.g., [30]).

The naive algorithm of checking all words is practically slow, as it does not involve any optimization and works always in time $\mathcal{O}^*(k^r)$. The standard algorithm, e.g., [23, 34, 42], for computing a shortest reset word is finding a path of state subsets in the power automaton (that is, the automaton whose set of states is 2^Q) from Q to a singleton. This works in at most $\mathcal{O}(kn \cdot 2^n)$ time but is practically faster as we usually traverse through fewer sets. Note that n may be much smaller than r (we know examples where r can be quadratic in n , e.g., [2]), but in the average case it is the opposite. The main drawback of this algorithm is its requirement of $\mathcal{O}(2^n)$ space, which is acceptable only up to small $n \sim 30$.

Alternative approaches include utilizing SAT solvers [38] by suitable reductions of the problem and binary search over possible values of r . SAT solvers were also recently tried for partial deterministic finite automata and *careful synchronization* [36]. In the reported results, such solutions reach random binary automata with about 100 states.

The fastest algorithm so far is based on a bidirectional search of the power automaton, equipped with several enhancements [19, 20]. This algorithm was able to deal with binary random automata up to 350 states. Despite several later attempts, no faster solutions were developed and the algorithm was not improved until now.

1.3.2 Heuristic Algorithms

The most classic polynomial algorithm is Eppstein’s one [10]. It works in $\mathcal{O}(n^3 + kn^2)$ time and finds a reset word of length at most $(n^3 - n)/3$ (due to the Frankl-Pin’s bound). Several heuristic improvements were proposed, e.g., Cycle, SynchroP, and SynchroPL algorithms [23, 42], which do not improve guarantees but behave better in experimental settings, even at the cost of increased worst-case time complexity. Recent works also involve attempts to speed-up heuristics by adapting to parallel and GPU computation [35, 41].

A remarkable heuristic is the *beam* algorithm based on inverse breadth-first search ([30, *CutOff-IBFS*]), i.e., starting from a singleton and ending with Q , which significantly beats other algorithms based on the forward search. Yet, curiously, it does not provide any worst-case guarantees, as it theoretically may not find any reset word at all.

Alternative approaches involve artificial intelligence methods, e.g., hierarchical classifier [28], genetic algorithms [22], and machine learning approaches [27].

1.4 Contribution

We reinvestigate the so-far best exact algorithm [19, 20] and significantly improve it. We develop a series of algorithmic enhancements involving better data structures, decision mechanisms, and reduction procedures. Altogether, we obtain a significant speed-up and decrease the memory requirements. Additionally, the remodeled algorithm is adapted for effective usage of multithreading and GPU computing, which was not possible in the original.

On the implementation side, we develop an open computational package containing the new exact algorithm as well as several known polynomial heuristics. We apply a series of technical optimizations and fine-tune the algorithm to maximize efficiency. The package supports configurable just-in-time compiled computation plans and can be extended with new algorithms.

In the experimental section, we test the efficiency of the algorithm and compute the reset thresholds of binary random automata up to 570 states. We refine the previous estimation formula for the expected reset threshold of these automata.

The computational package is available at [40] (the version related to this paper is 1.1.0).

2 The New Exact Algorithm

Our algorithm is based on the former best exact algorithm [19, 20]. While the new version differs in the choice of data structures and subprocedures, at a high level it is similar and uses two main phases – bidirectional breadth-first search and then inverse depth-first search.

The input to the algorithm is an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ with n states and k input letters. The goal is to find its reset threshold r . In the first step, we check if \mathcal{A} is synchronizing by the well-known procedure [44], which checks for every two states $p, q \in Q$ whether they can be mapped to one state; this is doable in $\mathcal{O}(kn^2)$. Then we get upper bounds on the reset threshold by using polynomial-time heuristics. For this, we use the Eppstein algorithm [10] at first, and then the enhanced *beam* algorithm [30, *CutOff-IBFS*]. The found upper bound helps the main procedure make better decisions.

Given a subset $S \subseteq Q$ and a word $w \in \Sigma^*$, the *image* of S under the action of w is $\delta(S, w) = \{\delta(q, w) \mid q \in S\}$. The *preimage* of S under the action of w is $\delta^{-1}(S, w) = \{q \in Q \mid \delta(q, w) \in S\}$.

The key idea is to simultaneously run a breadth-first search (BFS) starting from the set Q and computing images, together with an inverse breadth-first search (IBFS) starting from all of the singletons and computing preimages. While both algorithms on their own

require computation of at most k^r or at most nk^r sets respectively, combining them lets us compute no more than $nk^{r/2}$ sets, provided that we can somehow test if the searches have met. To do this, we need to check if there exists a pair X, Y of sets, belonging respectively to the BFS and IBFS lists, such that $X \subseteq Y$. Indeed, then we know that there are words $x, y \in \Sigma^*$ such that $X = \delta(Q, x)$ and $Y = \delta^{-1}(\{q\}, y)$ for some $q \in Q$. Because $X \subseteq Y$, we get $\delta(Q, xy) = \{q\}$, which means that xy is a reset word. Due to the *Orthogonal Vectors Conjecture* [16], there is probably no subquadratic solution to this subset problem¹. Such a solution would also contradict the mentioned fact that we cannot find the reset threshold in $\mathcal{O}^*((k - \varepsilon)^r)$ time assuming SETH. Nevertheless, we employ procedures that work well in our practical case. They are also used to reduce the number of sets in the lists during the searches, which effectively lowers the branching factor. Finally, we do not actually run the two searches until they meet. Instead, we switch to the second phase with inverse DFS (which takes the steps only on the IBFS side computing preimages), when either the memory runs out or we calculate that it should be faster based on the upper bound from the heuristics and the collected statistics.

These high-level ideas are derived from the previous algorithm. However, we design different, more efficient procedures and optimizations for these steps so that it is possible to solve the problem significantly faster and for larger automata. First, we modify data structures and redesign how a single iteration of BFS / IBFS works. Apart from making the bidirectional-search phase faster, it allows completing more iterations before switching to the DFS phase due to the lower memory consumption, which is crucial in the case of large automata. The decision-making part of the algorithm is also extended. We use five types of steps, and the decision on which step to take is based on statistics from the current algorithm's run and forward prediction. In the DFS phase, we enhance the radix trie data structure in terms of both efficiency and memory overhead. We also apply some forms of list reductions, which decrease the branching factor. Finally, every part of the new algorithm can be parallelized in one way or the other, which was not possible before; the general difficulty of parallelization comes from large shared data structures and a lot of branching.

In the next sections, we describe the new algorithmic techniques. For the sake of brevity, we consider a version that only calculates the reset threshold. The algorithm can be trivially modified to also return the reset word by storing pointers to predecessors along with the sets, although then either time or memory footprint is slightly increased.

2.1 Bidirectional Breadth-First Search

The first phase of the algorithm consists of running the two breadth-first searches. The BFS starts with a list L_{BFS} containing just the set Q . When the search starts a new iteration, L_{BFS} is replaced with $\{\delta(S, a) \mid S \in L_{\text{BFS}}, a \in \Sigma\}$. Conversely, L_{IBFS} is initialized with all the singletons and the list is replaced with $\{\delta^{-1}(S, w) \mid S \in L_{\text{IBFS}}, w \in \Sigma\}$.

We say that the two searches *meet* if there exist $X \in L_{\text{BFS}}$ and $Y \in L_{\text{IBFS}}$ for which $X \subseteq Y$ holds. The meet condition implies that the lists can be reduced by removing the elements which are not minimal (and respectively maximal for IBFS) with respect to inclusion. We can reduce the lists further by ensuring that no new set is a superset (subset for IBFS) of a set belonging to some list from any previous iteration. To make this possible, we keep track

¹ The *Orthogonal Vectors problem* gives two sets A, B of Boolean vectors of the same length and asks if there exists a pair $(u \in A, v \in B)$ such that u and v are orthogonal, i.e., $u \cdot v = \mathbf{0}$. We can reduce our problem to *OV* by transforming the sets in Y to their complements and then representing all the sets as their characteristic vectors.

of all the visited sets in two additional *history* lists H_{BFS} , H_{IBFS} . This reduction, although usually helpful during most of the iterations, at the end may turn out to be unprofitable, in which case the algorithm will drop the history list(s).

The original idea for the subprocedure to check the meet condition was to keep the lists as dynamic *radix tries*, supporting insertion and subset (or superset) checking operations. Now, instead, we take a somewhat simpler approach and operate directly on the lists, stored as random access containers (such as vectors in C++). We call this subprocedure $\text{MarkSupersets}(A, B)$ (and a similar one – $\text{MarkProperSupersets}(A, B)$, which additionally restricts the marked supersets to be non-equal to their subsets).

We split the reductions into three subprocedures: removing duplicates, self-reduction, and then reduction by history. In contrast to performing only one and the most expensive reduction by history (which could also include the first two reductions), after each subprocedure the list size gets smaller, which makes the next one run faster.

Alg. 1 shows the pseudocode of the bidirectional-search phase.

2.1.1 Subset and Superset Checking

There exist several algorithms solving the extremal sets problem in practical settings, e.g., [3]. They take a list of sets and mark all those that are not a subset of any other set.

We use a similar method to those utilizing a lexicographic sort, but we operate on two lists and are allowed to change the order of the second list during computation. MarkSupersets (Alg. 2) takes lists A and B and swaps sets in B so that those sets that are supersets of some sets from A appear at the end. The sets are treated like binary strings, i.e., their characteristic vectors, of length n . The procedure recursively splits the sets in A into those containing the d -th state and those not containing it, where d is the recursion depth. In this sense, it works by implicitly building a radix trie on A . We require that A is sorted lexicographically and its elements are unique, which can be guaranteed relatively cheaply before calling the procedure, as sorting is much faster than subset checking. The order gives us the property that the sets containing the d -th state and those not containing it are stored in continuous segments, so we can effectively split A . This lets us simulate in-place trie traversal with a recursive procedure that takes intervals of the lists as inputs. When the intervals are small (determined by the constant parameter MIN , Alg. 2 line 2), we can use a brute-force check instead of recursing further, which makes the procedure faster (especially important with GPU).

MarkSupersets is used to reduce the BFS list and to check the meet condition. To implement the MarkSubsets procedure needed on the IBFS side, we simply convert the sets to their complements and call MarkSupersets . The procedure $\text{MarkProperSupersets}$ is identical except for checking the containment for a pair of sets, where we additionally check that the two sets are different. When we use multithreading, we split the B list into equal parts after shuffling and execute parallel calls of the procedure. On GPU, we increase the MIN parameter and run the brute-force part there.

2.1.2 Decision Procedure

As the algorithm progresses, some steps may become unprofitable. The history lists, though helpful at the beginning, increase memory usage and cause a slow down if used in late iterations. Similarly, list reductions via MarkSupersets decrease the branching factor, but they are not that crucial when the search is approaching the reset threshold upper bound.

Algorithm 1 Bidirectional breadth-first search.

Input: A synchronizing automaton $\mathcal{A} = (Q, \Sigma, \delta)$ with $n = |Q|$ states and $k = |\Sigma|$ input letters. An upper bound R on the reset threshold.

Output: Reset threshold r .

```

1:  $L_{\text{BFS}}, H_{\text{BFS}} \leftarrow \{Q\}$ 
2:  $L_{\text{IBFS}}, H_{\text{IBFS}} \leftarrow \{\{q\} \mid q \in Q\}$ 
3: for  $r$  from 1 to  $R - 1$  do
4:   switch  $\text{CalculateBestStep}()$  do
5:     case BFS
6:       if  $H_{\text{BFS}}$  has grown significantly since its last reduction then
7:         Delete subsets from  $H_{\text{BFS}}$  that are larger than the largest ones from  $L_{\text{BFS}}$ 
8:         Delete non-minimal subsets from  $H_{\text{BFS}}$  ( $\text{MarkProperSupersets}$ )
9:       end if
10:       $L_{\text{BFS}} \leftarrow \text{CalculateImages}(L_{\text{BFS}})$ 
11:      Delete duplicates from  $L_{\text{BFS}}$  (lex. sort)
12:      Delete non-minimal subsets from  $L_{\text{BFS}}$  ( $\text{MarkProperSupersets}$ )
13:      Delete supersets of  $H_{\text{BFS}}$  from  $L_{\text{BFS}}$  ( $\text{MarkSupersets}$ )
14:       $H_{\text{BFS}} \leftarrow H_{\text{BFS}} \cup L_{\text{BFS}}$ 
15:     case  $\text{BFS}^{\text{NH}}$  (without history)
16:        $L_{\text{BFS}} \leftarrow \text{CalculateImages}(L_{\text{BFS}})$ 
17:       Delete duplicates from  $L_{\text{BFS}}$  (lex. sort)
18:       Delete non-minimal subsets from  $L_{\text{BFS}}$  ( $\text{MarkProperSupersets}$ )
19:     case IBFS
20:       ... ▷ Analogous to BFS
21:     case  $\text{IBFS}^{\text{NH}}$  (without history)
22:       ... ▷ Analogous to BFS (without history)
23:     case DFS
24:        $\text{DFS}(\text{BuildStaticTrie}(L_{\text{BFS}}), L_{\text{IBFS}}, r, R)$ 
25:       return  $R$  ▷ DFS sets  $R \leftarrow$  the reset threshold
26:   if  $\text{MarkSupersets}(L_{\text{BFS}}, L_{\text{IBFS}})$  has found at least one superset then
27:     return  $r$ 
28:   end if
29: end for
30: return  $R$ 

```

We distinguish five types of steps from which the algorithm always chooses one for the next iteration – DFS, BFS, IBFS, BFS without the history list (denoted by BFS^{NH}) and IBFS without the history list (denoted by IBFS^{NH}). To assess which option to choose, we roughly estimate the cost subset checking operations each of them will require.

We reuse some of the equations previously defined in [20]. In particular, under simplifying assumptions about the uniform distribution of the states in sets we take their upper bound from [20, Theorem 4] previously applied to tries. Since our procedure can be interpreted as building a trie implicitly on the fly, this bound can also serve as a rough bound on the expected number of subset checking operations in a call to MarkSupersets . Let A_s, B_s be the size of the lists and A_d, B_d be their densities, i.e., for a list L let $\text{density}(L) = \frac{\sum_{s \in L} |S|}{n|L|}$.

Algorithm 2 Recursive procedure *MarkSupersets*.

Input: Lexicographically sorted list intervals A and B of sets with unique elements. Current depth of recursion d ($d = 0$ for the initial call).

```

1: procedure MARKSUPERSETS( $A, B, d$ )
2:   if  $|A| < MIN$  then                                     ▷ Brute-force for small  $|A|$ 
3:     Check each pair in  $A \times B$  and delete the supersets from  $B$  (move at the end and
       shrink the interval).
4:   else
5:      $A_0, A_1 \leftarrow A$  split by the  $d$ -th bit           ▷  $A$  is sorted, so a binary search suffices
6:     MarkSupersets( $A_0, B, d + 1$ )
7:     Sort  $B$  by the  $d$ -th bit.                               ▷ Linear time scan
8:      $B_1 \leftarrow$  interval in which the  $d$ -th bit is set   ▷ Suffix of  $B$ 
9:     MarkSupersets( $A_1, B_1, d + 1$ )
10:  end if
11: end procedure

```

Then, analogously to [20, EXPNVN in 4.2], we define:

$$ExpMark(A_s, B_s, A_d, B_d) = B_s \left(\frac{1 + B_d}{B_d} + \frac{1}{A_d - A_d B_d} \right) A_s^{\log_w(1 + B_d)},$$

where $w = (1 + B_d)/(1 + A_d B_d - A_d)$.

To estimate the sizes of the lists after (and in between) the reductions, we store the ratios $r_{\text{BFS}}^{\text{dupl}}, r_{\text{BFS}}^{\text{self}}, r_{\text{BFS}}^{\text{hist}}$ of the reduced sets respectively by the removal of duplicates, the removal of non-minimal subsets, and the reduction by history list. For instance, $r_{\text{BFS}}^{\text{dupl}}$ is the fraction of the removed duplicates during the first reduction. Similarly, separate ones are stored for the IBFS counterpart. In addition to the cost of subset checking, we also add the cost of computing sets themselves and reduction of duplicates (the constant *SetCost*, set to 512 in the implementation); however, in most cases, the cost of subset checking is dominant.

For instance, the cost of the next BFS step is calculated as follows:

$$BFS_{\text{cost}} = SetCost \cdot k \cdot |L_{\text{BFS}}| \tag{1}$$

$$+ ExpMark(k \cdot (1 - r_{\text{BFS}}^{\text{dupl}}) \cdot |L_{\text{BFS}}|, \quad k \cdot (1 - r_{\text{BFS}}^{\text{dupl}}) \cdot |L_{\text{BFS}}|, \tag{2}$$

$$density(L_{\text{BFS}}), \quad density(L_{\text{BFS}}))$$

$$+ ExpMark(|H_{\text{BFS}}|, \quad k \cdot (1 - r_{\text{BFS}}^{\text{dupl}}) \cdot (1 - r_{\text{BFS}}^{\text{self}}) \cdot |L_{\text{BFS}}|, \tag{3}$$

$$density(H_{\text{BFS}}), \quad density(L_{\text{BFS}}))$$

$$+ ExpMark(k \cdot (1 - r_{\text{BFS}}^{\text{dupl}}) \cdot (1 - r_{\text{BFS}}^{\text{self}}) \cdot (1 - r_{\text{BFS}}^{\text{hist}}) \cdot |L_{\text{BFS}}|, \quad |L_{\text{IBFS}}|, \tag{4}$$

$$density(L_{\text{BFS}}), \quad density(L_{\text{IBFS}})).$$

The formula is the sum of the four costs: the set cost (1), which estimates the cost of computing the successors' list with images and reduction of duplicates, the self-reduction cost (2), which assumes that the list size was already reduced by the factor $r_{\text{BFS}}^{\text{dupl}}$, the reduction by history cost (3), which assumes both preceding reductions, and the meet condition check cost (4).

The formulas for BFS^{NH} , IBFS , and IBFS^{NH} are analogous. Additionally, if we cannot perform a step because there is not enough memory, we set its expected cost to ∞ . Once we choose BFS^{NH} , we free the history and no longer consider the BFS step with it, so in this case, we also set its cost to ∞ (this is symmetrical for IBFS^{NH} and IBFS).

Next, we try to predict the full costs of choosing these steps by estimating the number of operations under the assumption that the algorithm will transition into the DFS phase one iteration later (or in the current iteration in the case of the *DFS* option). First, we calculate the expected branching factor in the DFS phase

$$f = k \cdot (1 - r_{\text{IBFS}}^{\text{dupl}} \cdot \text{DFSReductionOfReduction}),$$

where we take the average reduction of duplicates from the IBFS step and reduce it by some factor for a more pessimistic estimation ($\text{DFSReductionOfReduction} = 1/k$ in the implementation). $r_{\text{IBFS}}^{\text{dupl}}$ is the ratio of duplicates removed in the IBFS list during the latest reduction (since the DFS also removes duplicates). We assume conservatively that the reset threshold is equal to the known upper bound R and from that, we get the estimated number of iterations $R - r$ that still need to be done, where r is the number of the current iteration, which just begins. The predicted full cost of the BFS option is as follows (the number of sets multiplied by the set computing costs together with the meet condition cost):

$$\begin{aligned} \text{DFS}_{\text{pred}} = & f \cdot \frac{f^{R-r} - 1}{f - 1} \cdot (\text{SetCost} \cdot \text{DFSSetCostWeight} \cdot k/f \\ & + \text{DFSCheckCostWeight} \cdot \text{ExpMark}(|L_{\text{BFS}}|, |L_{\text{IBFS}}|, \text{density}(L_{\text{BFS}}), \text{density}(L_{\text{IBFS}}))), \end{aligned}$$

where DFSSetCostWeight and $\text{DFSCheckCostWeight}$ are constants (both set to 0.25 in the implementation) compensating for the fact that the operations are faster in the DFS phase, as additional optimizations are possible.

Finally, the step with the lowest predicted full cost is chosen. As an exception to this rule, if we do not expect to switch into the DFS phase soon (the steps without the history have both larger costs than the regular steps), instead of comparing the prediction costs of every choice, we consider only the BFS and IBFS costs and greedily choose the one with the lower single-step cost. This makes the bidirectional search more balanced in the short term. Otherwise, BFS is strongly preferred due to the assumption that the rest of the steps will be (inverse) DFS, which computes preimages, but the early statistics are less relevant for its estimation.

2.1.3 Heuristic Upper Bound

Having a good upper bound R on the reset threshold, preferably tight, is crucial for making the right decisions, i.e., not giving away history or entering the DFS phase too late. On the other hand, we do not want to spend too much time computing the bound.

The beam algorithm has its parameter *beam size*, which limits the size of the list and thus directly controls the quality (i.e., how close R is to the reset threshold) and complexity trade-off. Instead of setting the beam size to be a function of n ([20, 30]), we use an adaptive approach. We run it first with a relatively small beam size to find some reasonable bound, and then use this bound to calculate a rough estimation of the running time of the exact algorithm. The beam size is selected so that the beam algorithm's cost is a small fraction of that of the exact algorithm. In practice, this means that the beam size is larger for automata with larger (upper bounds on) reset thresholds than for those automata with smaller ones, even when n is the same.

2.2 Depth-First Search

In the second phase, the algorithm switches to an inverse depth-first search, which allows staying within the memory limit by adjusting the maximum list size. During this phase, the steps are taken only on the IBFS side. The fact that the BFS list no longer changes allows the meet condition check to be optimized.

2.2.1 Static Radix Trie

The L_{BFS} list is stored in an optimized data structure that supports the *ContainsSubset* operation. It is based on a radix trie, in which the characteristic vectors of the sets are stored. The queries rely on traversing the trie and, in each step, descending to either both children or just the left (zero) child, depending on the queried set.

In contrast with the usual radix trie, we apply a few specific optimizations:

Variable state ordering. In each node at a depth d , instead of splitting the subtrees by the d -th state, we split by a state x chosen specifically for the node. The state x is chosen so that the number of sets stored in the current subtree that also contain x is the largest possible. In practice, this makes the queries faster, since, in vertices such that x does not belong to the queried set, a large number of sets is immediately skipped.

This optimization also implies path compression, as we do not have nodes that do not split the sets into two non-empty parts. To ensure this, we also exclude the case that x is contained in all the sets and do not use it as a division state.

Leaf threshold. For a fixed constant parameter MIN ($= 10$ in the implementation), when the number of sets in a subtree is less or equal to MIN , we store them all in one vertex and do not recurse further. This does not increase running time and lowers the memory overhead. Technically, we can already store the sets in a node whose left (zero) subtree contains at most MIN sets, which avoids creating an additional node.

Joint queries. Instead of checking each subset separately, we group them by the same cardinality and check one group in a call using the swapping technique as in Alg. 2. Grouping saves operations responsible for traversing the trie, and additionally, grouping by cardinality allows to make cheap size elimination checks as below.

Size elimination. Every node v stores the minimal size m_v of the sets in its subtree. When we query for subsets of size s , if $m_v \leq s$ does not hold, we do not recurse into the subtree of v . This enhancement is derived from the previous algorithm [20], yet due to joint queries, we make these checks cheaper by executing at most one such check in a node for one cardinality. A similar optimization from the previous algorithm is mask elimination – checking if the intersection of all subsets in the subtree is contained in the queries set, yet we do not use it as it does not improve performance in our case.

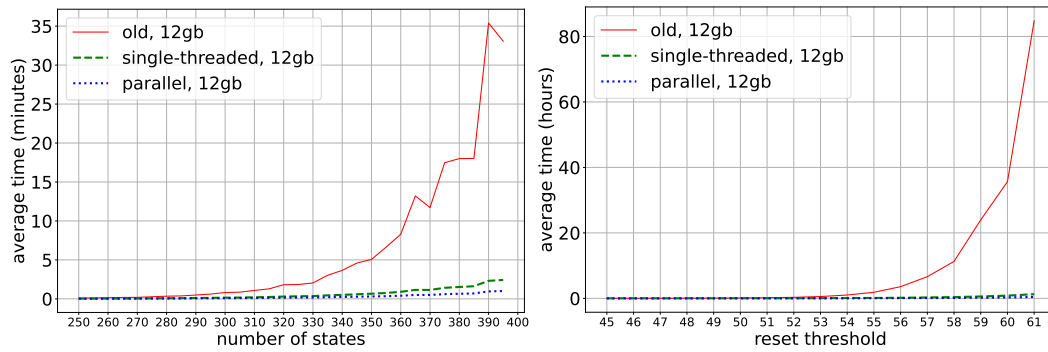
Our trie can be built in time $\mathcal{O}(|L_{\text{BFS}}| \cdot n^2)$ ($< n$ levels processed in time $\mathcal{O}(|L_{\text{BFS}}| \cdot n)$).

2.2.2 DFS Procedure

During the search, at each depth, the current list is split into parts of size at most $\text{available_memory}/((k+1)(R-r))$, where $R-r$ is the upper bound on the remaining number of steps to be done. Then, it recurses with each of these parts one by one, to make sure we do not run out of memory. The elements are sorted in order of descending cardinality so that the most promising sets are recursed first, which in turn can quickly improve the upper bound if it was not tight. The cardinality sort is also necessary for joint queries. The lists are reduced by the removal of duplicates and calls to *MarkSubsets* only once every few iterations, which still lowers the branching factor significantly. Parallel computation is performed by a thread pool with tasks being these separate calls for each cardinality.

3 Experiments

The implementation used for experiments is available at [40]. The tests were using `exact_reduce` configuration. The *old* algorithm [20] was run with the original code provided to us by the authors. The experiments with time measurement were run on computers with



■ **Figure 1** The mean running time for random binary automata with different numbers of states (left) and with different reset thresholds (right).

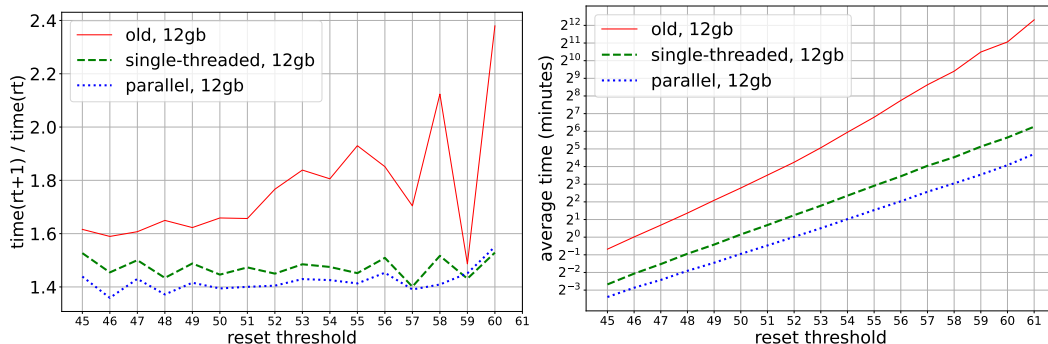
AMD Ryzen Threadripper 3960X 24-Core Processor, 64GB RAM, and two RTX3080 Nvidia GPU cards. We compiled the code using `gcc 9.3.0` and `nvcc 10.1` (run with `gcc 7.5.0`). We have tested the algorithm in both the *single-threaded* (only one thread without GPU) and *parallel* modes (6 threads and GPU enabled).

A *random* automaton with n states and k letters is generated by choosing each transition $\delta(q, a) \in Q$ uniformly at random, for $q \in Q, a \in \Sigma$. There is a negligible number of non-synchronizing automata obtained in this way, which were excluded (cf. [5]).

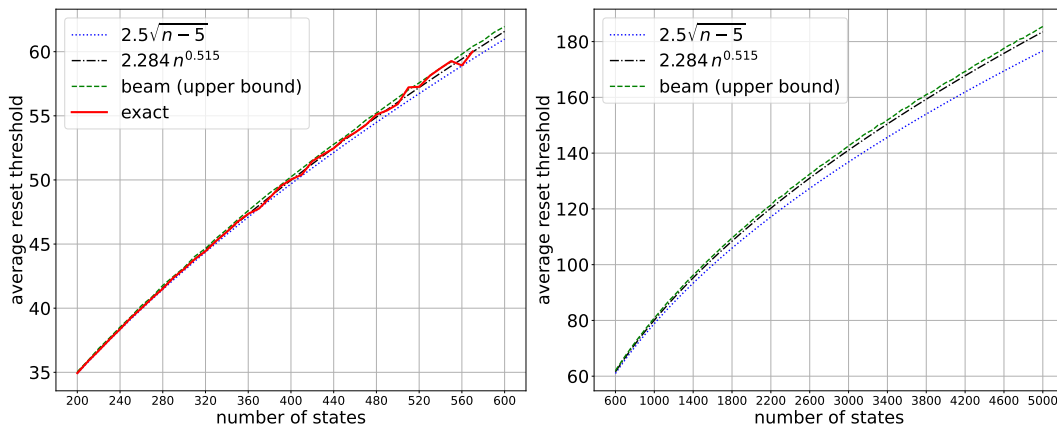
3.1 The Efficiency

Fig. 1 shows the efficiency comparison of our algorithm in both single-threaded and parallel modes, together with the old algorithm. This experiment was run for 1,000 random binary automata for each $n \in \{250, 255, \dots, 395\}$. We managed to test automata with up to 395 with the old algorithm, which took 3,177h computation time of a single process in total (we have used more memory, better hardware, and computed 10 times fewer automata per n than in the original experiments [20], which were done up to $n = 350$). The (unfinished) attempt to compute 1,000 automata with $n = 400$ by the old algorithm took over 770h, whereas our algorithm (single-threaded) finished in 48h. The total computation time of our algorithm up to $n = 395$ was resp. 299h in the single-threaded mode and 135h in the parallel mode.

Fig. 2 shows the average increase in running time when the reset threshold grows. The general observed tendency is a factor of about 1.5 for our algorithm.



■ **Figure 2** The mean running time growth factor in relation to reset threshold (left) and the mean running time in a logarithmic scale (right).



■ **Figure 3** The mean reset threshold of binary automata with n states. For every $n \in \{5, 10, \dots, 300\}$, $n \in \{305, 310, \dots, 400, 410, 420, \dots, 500\}$, and $n \in \{510, 520, \dots, 570\}$ we calculated resp. 10,000, 1,000, and 100 automata.

Hard instances

Most automata have their reset thresholds sublinear, but there exist other examples (though they are rare). As the reset threshold is the main indicator of difficulty, even instances with a small number of states should be difficult for algorithms.

From the known constructions, the most extreme automata with respect to the reset threshold are *slowly synchronizing* ones [2]. They have reset thresholds close to $(n-1)^2$, and the Černý series meets this bound. Yet, they all have the property that the IBFS list reduces to a constant number of sets in every iteration, thus our algorithm works in polynomial time for them, just as the old algorithm. An example of extreme automata without this property could be the unique series of automata with a *sink* state (i.e., a state $q \in Q$ such that $\delta(q, a) = q$ for all $a \in \Sigma$), that reaches the maximum reset threshold $n(n-1)/2$ in this class [31]. (A synchronizing automaton can have at most one sink state and a reset word must map all the states to it.) A *slowly sink* automaton from this series with 26 states has 25 letters and its reset threshold 325 is computed by the old algorithm in 34m 14s, whereas the new algorithm computes it in 7m 22s (parallel).

3.2 Mean reset threshold

In the second experiment, we computed reset thresholds of random binary automata with $n \in \{410, 420, \dots, 570\}$ states with our algorithm in parallel mode. The mean computation time for $n = 500$ was 9m 42s. Fig. 3 shows the mean reset threshold. In addition, up to $n = 5,000$ we computed a good upper bound using the beam algorithm with beam size $n \log n$ (we made the beam algorithm much faster due to GPU computation; the previous such experiments were done up to $n = 1000$ [30]).

It is now visible that the previous formula $2.5\sqrt{n-5}$ is underestimated. On the other hand, a standard approach² of deriving a function of the form $a(n+b)^c + d$ yields a wrong formula, lately exceeding the upper bound obtained by the beam. We decrease the exponent to fit with a more accurate estimation $2.284n^{0.515}$.

² We use the algorithm from `scipy.optimize.curve_fit` with the Levenberg-Marquardt algorithm.

4 Conclusions

We have improved the best-known algorithm for computing the (length of the) shortest reset words. While the overall idea of employing bidirectional breadth-first search is the same, we replace each of its subprocedures with more efficient ones.

The algorithm can be easily adapted to alternative synchronization settings and other related problems. For instance, it is trivial to use it for *careful* synchronization [36], where some transitions can be forbidden. It can also be adapted to, e.g, non-careful settings [6], *mortal words* [32], or subset synchronization [46].

For future work, we plan to use this new algorithmic tool to perform more extensive experiments concerning reset thresholds, especially with larger automata and with a larger alphabet. Finally, it can be used to experimentally verify or extend the current verification range of certain conjectures.


References

- 1 D. S. Ananichev and M. V. Volkov. Synchronizing monotonic automata. In *Developments in Language Theory*, volume 2710 of *LNCS*, pages 111–121. Springer, 2003.
- 2 D. S. Ananichev, M. V. Volkov, and V. V. Gusev. Primitive digraphs with large exponents and slowly synchronizing automata. *Journal of Mathematical Sciences*, 192(3):263–278, 2013.
- 3 R. J. Bayardo and B. Panda. *Fast Algorithms for Finding Extremal Sets*, pages 25–34. SIAM, 2011.
- 4 M. Berlinkov and M. Szykuła. Algebraic synchronization criterion and computing reset words. *Information Sciences*, 369:718–730, 2016.
- 5 M. V. Berlinkov. On the Probability of Being Synchronizable. In *Proceedings of the Second International Conference on Algorithms and Discrete Applied Mathematics - Volume 9602*, volume 9602 of *CALDAM*, pages 73–84. Springer, 2016.
- 6 M. V. Berlinkov, R. Ferens, A. Ryzhikov, and M. Szykuła. Synchronizing Strongly Connected Partial DFAs. In *STACS*, volume 187 of *LIPICs*, pages 12:1–12:16. Schloss Dagstuhl, 2021.
- 7 J. Berstel, D. Perrin, and C. Reutenauer. *Codes and Automata*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2009.
- 8 J. Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied*, 14(3):208–216, 1964. In Slovak.
- 9 M. de Bondt, H. Don, and H. Zantema. Lower Bounds for Synchronizing Word Lengths in Partial Automata. *Int. J. Found. Comput. Sci.*, 30(1):29–60, 2019.
- 10 D. Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19:500–510, 1990.
- 11 H. Fernau, P. Heggernes, and Y. Villanger. A multi-parameter analysis of hard problems on deterministic finite automata. *Journal of Computer and System Sciences*, 81(4):747–765, 2015.
- 12 P. Gawrychowski and D. Straszak. Strong inapproximability of the shortest reset word. In *Mathematical Foundations of Computer Science*, volume 9234 of *LNCS*, pages 243–255. Springer, 2015.
- 13 M. Gerbush and B. Heeringa. Approximating minimum reset sequences. In *Implementation and Application of Automata*, volume 6482 of *LNCS*, pages 154–162. Springer, 2011.
- 14 B. Gerencsér, V. V. Gusev, and R. M. Jungers. Primitive Sets of Nonnegative Matrices and Synchronizing Automata. *SIAM J. Matrix Anal. Appl.*, 39(1):83–98, 2018.
- 15 H. Jürgensen. Synchronization. *Information and Computation*, 206(9-10):1033–1044, 2008.
- 16 D. M. Kane and R. R. Williams. The Orthogonal Vectors Conjecture for Branching Programs and Formulas. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPICs)*, pages 48:1–48:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

- 17 J. Kari. Synchronization and stability of finite automata. *Journal of Universal Computer Science*, 8(2):270–277, 2002.
- 18 J. Kari and M. V. Volkov. Černý conjecture and the road colouring problem. In *Handbook of automata*, volume 1, pages 525–565. European Mathematical Society Publishing House, 2021.
- 19 A. Kisielewicz, J. Kowalski, and M. Szykuła. A Fast Algorithm Finding the Shortest Reset Words. In *COCOON*, volume 7936 of *LNCS*, pages 182–196, 2013.
- 20 A. Kisielewicz, J. Kowalski, and M. Szykuła. Computing the shortest reset words of synchronizing automata. *Journal of Combinatorial Optimization*, 29(1):88–124, 2015.
- 21 A. Kisielewicz, J. Kowalski, and M. Szykuła. Experiments with Synchronizing Automata. In *Implementation and Application of Automata*, volume 9705 of *LNCS*, pages 176–188. Springer, 2016.
- 22 J. Kowalski and A. Roman. A new evolutionary algorithm for synchronization. In Giovanni Squillero and Kevin Sim, editors, *Applications of Evolutionary Computation*, pages 620–635. Springer, 2017.
- 23 R. Kudłacik, A. Roman, and H. Wagner. Effective synchronizing algorithms. *Expert Systems with Applications*, 39(14):11746–11757, 2012.
- 24 C. Nicaud. Fast Synchronization of Random Automata. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *APPROX/RANDOM 2016*, volume 60 of *LIPICs*, pages 43:1–43:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.
- 25 J. Olschewski and M. Ummels. The complexity of finding reset words in finite automata. In *Mathematical Foundations of Computer Science 2010*, volume 6281 of *LNCS*, pages 568–579. Springer, 2010.
- 26 J.-E. Pin. On two combinatorial problems arising from automata theory. In *Proceedings of the International Colloquium on Graph Theory and Combinatorics*, volume 75 of *North-Holland Mathematics Studies*, pages 535–548, 1983.
- 27 I. Podolak, A. Roman, M. Szykuła, and B. Zieliński. A machine learning approach to synchronization of automata. *Expert Systems with Applications*, 97:357–371, 2018.
- 28 I. T. Podolak, A. Roman, and D. Jędrzejczyk. Application of hierarchical classifier to minimal synchronizing word problem. In *Artificial Intelligence and Soft Computing*, volume 7267 of *LNCS*, pages 421–429. Springer, 2012.
- 29 I. Pomeranz and S.M. Reddy. On achieving complete testability of synchronous sequential circuits with synchronizing sequences. *IEEE Proc. International Test Conference*, pages 1007–1016, 1994.
- 30 A. Roman and M. Szykuła. Forward and backward synchronizing algorithms. *Expert Systems with Applications*, 42(24):9512–9527, 2015.
- 31 I. K. Rystsov. Reset words for commutative and solvable automata. *Theoretical Computer Science*, 172(1-2):273–279, 1997.
- 32 A. Ryzhikov. Mortality and Synchronization of Unambiguous Finite Automata. In Robert Mercas and Daniel Reidenbach, editors, *Combinatorics on Words*, pages 299–311. Springer International Publishing, 2019.
- 33 A. Ryzhikov and M. Szykuła. Finding Short Synchronizing Words for Prefix Codes. In *MFCS 2018*, volume 117 of *LIPICs*, pages 21:1–21:14. Schloss Dagstuhl, 2018.
- 34 S. Sandberg. Homing and synchronizing sequences. In *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*, pages 5–33. Springer, 2005.
- 35 N. E. Saraç, O. F. Altun, K. T. Atam, S. Karahoda, K. Kaya, and H. Yenigün. Boosting expensive synchronizing heuristics. *Expert Systems with Applications*, 167:114203, 2021.
- 36 H. Shabana. Exact synchronization in partial deterministic automata. *Journal of Physics: Conference Series*, 1352:012047, 2019.
- 37 Y. Shitov. An Improvement to a Recent Upper Bound for Synchronizing Words of Finite Automata. *Journal of Automata, Languages and Combinatorics*, 24(2–4):367–373, 2019.

- 38 E. Skvortsov and E. Tipikin. Experimental study of the shortest reset word of random automata. In *Implementation and Application of Automata*, volume 6807 of *LNCS*, pages 290–298. Springer, 2011.
- 39 M. Szykuła. Improving the Upper Bound on the Length of the Shortest Reset Word. In *STACS 2018, LIPIcs*, pages 56:1–56:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.
- 40 M. Szykuła and A. Zyzik. Synchronwords. <https://github.com/marekesz/synchronwords>, 2022.
- 41 M. K. Taş, K. Kaya, and H. Yenigün. Synchronizing billion-scale automata. *Information Sciences*, 574:162–175, 2021.
- 42 A. N. Trahtman. An efficient algorithm finds noticeable trends and examples concerning the Černý conjecture. In *Mathematical Foundations of Computer Science*, volume 4162 of *LNCS*, pages 789–800. Springer, 2006.
- 43 N. F. Travers and J. P. Crutchfield. Exact Synchronization for Finite-State Sources. *Journal of Statistical Physics*, 145(5):1181–1201, 2011.
- 44 M. Volkov. Synchronizing automata and the Černý conjecture. In *Language and Automata Theory and Applications*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008.
- 45 M. V. Volkov, editor. *Special Issue: Essays on the Černý Conjecture*, volume 24 (2–4) of *Journal of Automata, Languages and Combinatorics*, 2019.
- 46 V. Vorel. Subset Synchronization and Careful Synchronization of Binary Finite Automata. *International Journal of Foundations of Computer Science*, 27(05):557–577, 2016.
- 47 V. Vorel. Complexity of a problem concerning reset words for Eulerian binary automata. *Information and Computation*, 253:497–509, 2017.

Fast RSK Correspondence by Doubling Search

Alexander Tiskin 

Department of Mathematics and Computer Science, St. Petersburg State University, Russia

Abstract

The Robinson–Schensted–Knuth (RSK) correspondence is a fundamental concept in combinatorics and representation theory. It is defined as a certain bijection between permutations and pairs of Young tableaux of a given order. We consider the RSK correspondence as an algorithmic problem, along with the closely related k -chain problem. We give a simple, direct description of the symmetric RSK algorithm, which is implied by the k -chain algorithms of Viennot and of Felsner and Wernisch. We also show how the doubling search of Bentley and Yao can be used as a subroutine by the symmetric RSK algorithm, replacing the default binary search. Surprisingly, such a straightforward replacement improves the asymptotic worst-case running time for the RSK correspondence that has been best known since 1998. A similar improvement also holds for the average running time of RSK on uniformly random permutations.

2012 ACM Subject Classification Mathematics of computing → Permutations and combinations

Keywords and phrases combinatorics of permutations, Robinson–Schensted–Knuth correspondence, k -chains, RSK algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.86

Funding This work was supported by the Russian Science Foundation under grant no. 22-21-00669, <https://www.rscf.ru/en/project/22-21-00669/>.

Acknowledgements I thank Nikolay Vasilyev, Vasilii Duzhin and Artem Kuzmin for advice and fruitful discussions.

1 Introduction

The Robinson–Schensted–Knuth (RSK) correspondence is a fundamental concept in combinatorics and representation theory; for the background on the combinatorial aspects of RSK, see e.g. [20, 17]. It is defined as a certain bijection between pairs of standard Young tableaux and permutations of a given order, and represents a far-reaching generalisation of the longest increasing subsequence problem in a permutation. A common definition of RSK correspondence is algorithmic, via Robinson–Schensted tableau insertions or, alternatively, via the Viennot geometric construction.

The combinatorial properties of RSK are well-studied. In this paper, we consider the RSK correspondence as an algorithmic problem, along with the closely related k -chain problem. In particular, we are interested in both the worst-case and the average asymptotic running time of algorithms for these problems. This aspect of the RSK correspondence seems to have been studied relatively less thoroughly than its combinatorial aspects.

In the rest of this paper, we recall the definition of the RSK correspondence, using the geometric construction of Viennot [23, 24]. We then describe the standard RSK algorithm by Robinson [15] and Schensted [18]. Further, we give a simple, direct description of the symmetric RSK algorithm, which is implied by the k -chain algorithms of Viennot [24] and of Felsner and Wernisch [9]. Next, we recall the doubling search algorithm of Bentley and Yao [1], and show how it can be used as a subroutine by the symmetric RSK algorithm, replacing the default binary search. Surprisingly, such a straightforward replacement improves the asymptotic worst-case running time for the RSK correspondence from $O(n^{3/2} \log n)$, which has been the best known since [9], to $O(n^{3/2})$. A similar improvement also holds for the average running time of RSK on uniformly random permutations.



© Alexander Tiskin;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 86; pp. 86:1–86:10



Leibniz International Proceedings in Informatics

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

2 The RSK correspondence

Partial orders. We will use the standard terminology related to partial orders: *downset*, *principal downset*, *chain*, *antichain*. We consider two mutually inverse (strict) total orders on \mathbb{R} : $<$ and $>$. We also consider the corresponding (strict) partial *dominance* orders on \mathbb{R}^2 : \ll , \leq , \geq , \gg , where $(x, y) \ll (x', y')$ if $x < x'$ and $y < y'$, and similarly for the other three orders. Dominance orders \ll , \gg are mutually inverse, and so are \leq , \geq . When considering a point set P as a partial order, we will indicate it by a superscript, e.g. P^{\ll} . We will always assume that P is finite, and that all x -coordinates in P are distinct, and so are all the y -coordinates.

Young tableaux. Let \mathbb{N}_+ denote the set of all positive integers. Given $n \in \mathbb{N}_+$, let $\mathbb{N}_n = \{1, \dots, n\} \subset \mathbb{N}_+$.

► **Definition 1.** A Young diagram of order n is a subset of \mathbb{N}_+^2 of cardinality n , that is a downset in the dominance partial order \ll . A Young tableau¹ of order n is an order-preserving bijection from a Young diagram of order n (called the tableau's shape) to a subset of \mathbb{R} with total order $<$.

We use the so-called French notation for visual representation of Young diagrams and tableaux. The elements of a diagram are represented by cells of an integer grid, arranged in left-aligned rows and bottom-aligned columns. Columns are ordered from left to right, and rows from below upwards. The value of each cell of a tableau is written within that cell; these values increase from left to right in rows, and from below upwards in columns.

► **Example 2.** Figure 1 (middle and right columns) gives several examples of Young tableaux with cell values in \mathbb{N}_{10} .

Canonical antichain partitioning. The theory of Young tableaux is intimately connected with the combinatorics of permutations. We take a symmetric view of this connection, due to Viennot [23, 24]. A permutation, viewed as a mapping $\pi : \mathbb{N}_n \rightarrow \mathbb{N}_n$, is identified with the mapping's graph, i.e. the point set $P_\pi = \{(x, \pi(x)) \mid x \in \mathbb{N}_n\}$.

► **Definition 3.** The height of an element in a finite partial order O is the maximum cardinality of a chain in the principal downset generated by that element. A canonical antichain is formed by all the elements of a given height. The partitioning of O into disjoint canonical antichains is called the canonical antichain partition (CAP), denoted $\text{cap}(O)$.

Canonical antichains in \mathbb{R}^2 are also sometimes called *layers of minima (maxima)* [4, 3], *Pareto fronts* [5], or *terraces* [14]. The canonical antichain partition of a point set in \mathbb{R}^2 is also sometimes called *greedy cover* [12], *patience sorting* [2], or *non-dominated sorting* [5].

► **Example 4.** Figure 1 (top-left) shows a point set P of cardinality 10, and its partitioning $\text{cap}(P^{\ll})$ into five antichains.

We recall the following standard result.

► **Proposition 5.** The partitioning $\text{cap}(O)$ of a finite partial order has the minimum possible number of antichains among all antichain partitionings of O . This number is also equal to the maximum cardinality of a chain in O .

Proof. Straightforward; see e.g. [9]. ◀

¹ Young tableaux as defined here are often called “standard”, to distinguish them from more general types of tableaux; we omit this qualifier, since it is the only type of Young tableaux we are dealing with.

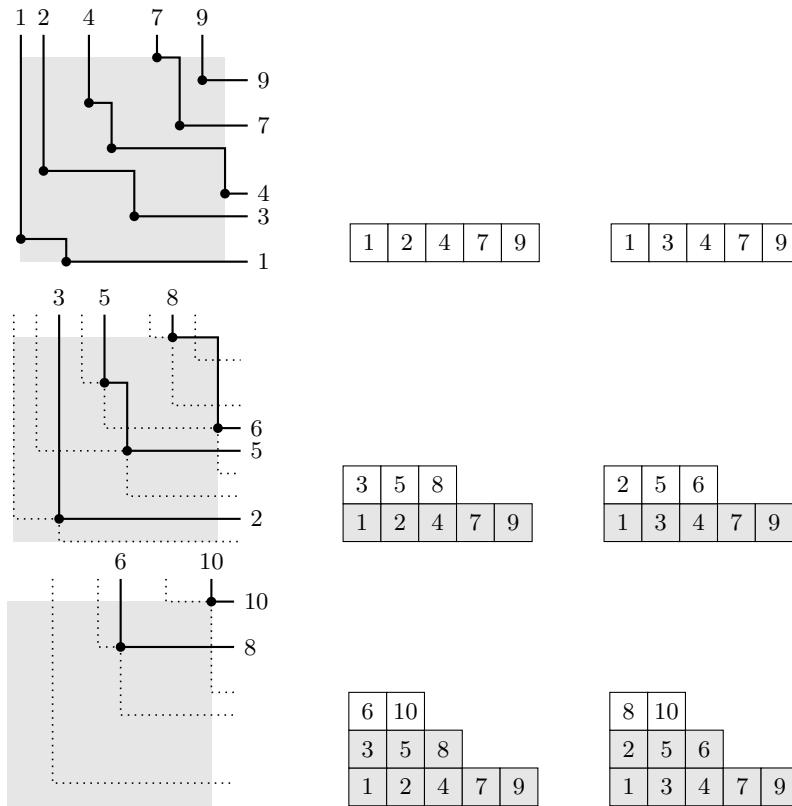


Figure 1 The standard RSK algorithm for $rsk(P^{\ll}) = (H, T)$; tableaux H, T obtained by rows.

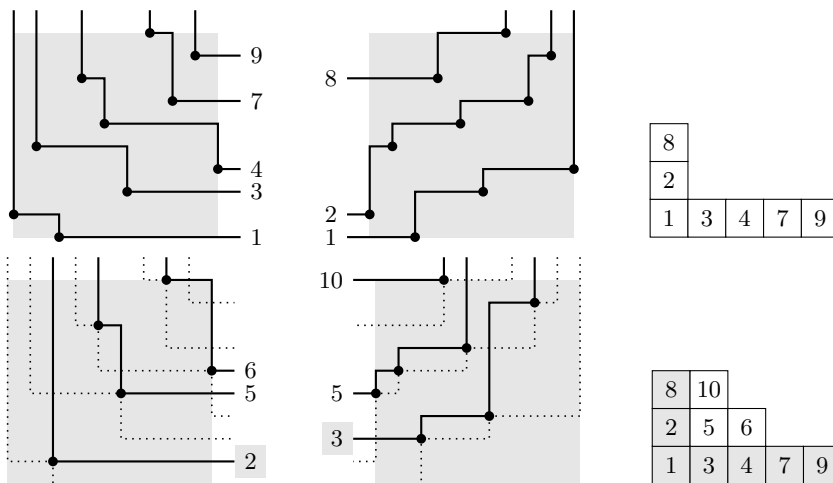


Figure 2 The symmetric RSK algorithm for $rsk(P^{\ll}) = (_, T)$ and $rsk(P^{\gg}) = (_, T^\dagger)$: tableau T obtained by principal hooks.

Given a permutation π , the problem of finding the cardinality of $\text{cap}(P_\pi^{\ll})$ is equivalent to the problem of finding the length of a longest increasing subsequence (LIS) in π . The LIS problem has a long history, going back to Erdős and Szekeres [8] and Robinson [15]. Based on their ideas, the classical LIS algorithm running in time $O(n \log n)$ was made explicit by Knuth [13], Fredman [10] and Dijkstra [6].

► **Definition 6.** Let P be a point set with dominance order \ll . Let $A = \{(x_1, y_1) \leq (x_2, y_2) \leq \dots \leq (x_r, y_r)\} \subseteq P$ be an antichain of cardinality $r \geq 1$. Values x_1 and y_r will be called respectively the head and the tail of A . The skeleton of A is the antichain $\text{sk}(A) = \{(x_2, y_1) \leq (x_3, y_2) \leq \dots \leq (x_r, y_{r-1})\}$ of cardinality $r - 1$. The skeleton of P is the point set $\text{sk}(P^{\ll}) = \bigcup_{A \in \text{cap}(P^{\ll})} \text{sk}(A)$. The heads, tails and skeletons with respect to dominance orders \leq, \geq, \gg are defined analogously.

The RSK correspondence. The Robinson–Schensted–Knuth (RSK) correspondence, discovered independently by Robinson [15] and Schensted [18] (see also Romik [17]), is a bijection between permutations of a given order and pairs of Young tableaux of the same order and identical shape. The two tableaux in the pair will be called the *head* and the *tail* tableaux (such a terminology is chosen for its symmetry and consistency with the rest of our exposition, whereas the traditional terminology calls them the *recording* and the *insertion* tableaux).

► **Definition 7.** Let P be a point set with dominance order \ll . The RSK image² of P is a pair of Young tableaux $\text{rsk}(P^{\ll}) = (H, T)$, defined recursively as follows. The initial row in H (respectively, T) is formed by the heads (respectively, the tails) of the antichains in $\text{cap}(P^{\ll})$. The remaining rows of H, T are formed as $\text{rsk}(\text{sk}(P^{\ll}))$. The RSK image with respect to dominance orders \leq, \geq, \gg is defined analogously.

► **Example 8.** Figure 1 (top row) shows the construction of the initial rows in tableaux $\text{rsk}(P^{\ll}) = (H, T)$ from a point set $P \subseteq \mathbb{N}_{10}^2$ (top left). Figure 1 (middle and bottom rows) shows the recursive construction of the remaining rows in the tableaux H, T .

The RSK correspondence has some beautiful symmetries, exposed by Schützenberger [19].

► **Definition 9.** Let P be a point set. Its transpose is the point set $P^\dagger = \{(x, y) \mid (y, x) \in P\}$ obtained by exchanging the x - and y -coordinates of every point.

► **Observation 10.** Let π be a permutation. We have $P_\pi^\dagger = P_{\pi^{-1}}$.

► **Definition 11.** Let Y be a Young diagram. Its transpose Y^\dagger is the Young diagram obtained by exchanging the x - and y -coordinates of every cell.

We now state the theorem by Schützenberger [19] on the symmetries of the RSK correspondence; for completeness, we also present its proof.

► **Theorem 12 (Schützenberger).** Let P be a point set, $\text{rsk}(P^{\ll}) = (H, T)$. We have

- (i) $\text{rsk}(P^{\dagger \ll}) = (T, H)$,
- (ii) $\text{rsk}(P^{\geq}) = (H^{*\dagger}, T^\dagger)$,
- (iii) $\text{rsk}(P^{\leq}) = (H^\dagger, T^{*\dagger})$,
- (iv) $\text{rsk}(P^{\gg}) = (H^*, T^*)$.

Here, H^*, T^* are Young diagrams of the same shape as H, T , called the Schützenberger dual of H, T , respectively.

² The terms “RSK correspondence”, “RSK image” as defined here are often called just “Robinson–Schensted”, reserving the name “RSK” for a more general type of combinatorial bijection. Since this is an algorithmic study, we use the term RSK throughout, in order to highlight the contribution of Donald Knuth to the development of RSK algorithms.

Proof. Part (i) is obvious by symmetry.

Let us establish part (ii). Let (x_0, y_0) be the point with the least y -coordinate in $sk(P^{\ll})$, so y_0 is the tail of the least-height antichain in $cap(sk(P^{\ll}))$. Let A be an antichain in $cap(P^{\ll})$ such that $(x_0, y_0) \in sk(A)$. Then, there is a pair of points $(x_0, y) \geq (x, y_0)$ in A . Let B be the subset of points in P with y -coordinate less than y_0 . Subset B must consist of a single chain including point (x_0, y) : otherwise, there would be two points with y -coordinate less than y_0 in some antichain A' of $cap(P^{\ll})$, and then $sk(A')$ would contain a point of $sk(P^{\ll})$ with y -coordinate less than y_0 , which would contradict the minimality of y_0 . Now consider the points of B with the partial order \geq ; these points, including point (x_0, y) , form a subset of the least height antichain in $cap(P^{\geq})$. Point (x, y_0) must belong to the second-least height antichain in $cap(P^{\geq})$, and must have the least y -coordinate in that antichain. Therefore, y_0 is the tail of the second-least height antichain in $cap(P^{\geq})$.

We have established that the tail of the least height antichain in $cap(sk(P^{\ll}))$ is equal to the tail of the second-least height antichain in $cap(P^{\geq})$. Now (ii) follows by (i) and the recursive construction of Definition 7, and (iii), (iv) follow from (ii) by symmetry. ◀

Multichains. A notion closely related to the subject of this paper is that of a k -chain.

► **Definition 13.** Let O be a finite partial order. A k -chain is a subset of O that can be represented as a union of k chains.

The connection between the RSK correspondence and k -chains is given by the classical theorem of Greene [11].

► **Theorem 14 (Greene).** Let P be a point set with dominance order \ll . The maximum cardinality of a k -chain in P is equal to the number of cells in the initial k rows of the shape of $rsk(P^{\ll})$.

In fact, the RSK algorithm by Felsner and Wernisch [9] is presented entirely in the language of k -chains. While we use the language of Young tableaux in this paper, our results translate immediately into the corresponding statements on maximum k -chains in a set of points, and thus relate to the results of [9].

3 RSK algorithms

Standard RSK algorithm. Definition 7 leads directly to the following standard algorithm for computing the RSK image of a given point set.

Given a point set P , the pair of tableaux $rsk(P^{\ll}) = (H, T)$ are constructed by rows. To obtain the initial rows of H, T , the points in P are scanned in order of increasing x -coordinate. For the subset Q of points seen so far, we maintain the partitioning $cap(Q^{\ll})$; in particular, the heads and the tails of antichains in that partitioning are kept in sorted order. We also maintain the skeleton $sk(Q^{\ll})$ in order of increasing x -coordinate. When the scan of P is complete ($Q = P$), the heads (respectively, tails) of antichains in $cap(P^{\ll})$ become the initial row of tableau H (respectively, T) in $rsk(P^{\ll})$. To obtain the remaining rows of $rsk(P^{\ll})$, we repeat the above procedure on point set $sk(P^{\ll})$. Algorithm 1 gives the algorithm's pseudocode.

► **Example 15.** Figure 1 shows the execution of the standard RSK algorithm in three successive iterations: the point set at the beginning of each iteration and its CAP (left column), and the state of the tableaux H and T at the end of the respective iteration (middle and right columns).

■ **Algorithm 1** Standard RSK. The choice of a search method in line 7 is either linear or binary (Section 3) or doubling (Section 4).

```

1: procedure RSK( $P$ )      ▷ given point set  $P$  sorted by  $x$ -coordinate, returns  $rsk(P^{\ll})$ 
2:   if  $P = \emptyset$  then return  $(\emptyset, \emptyset)$ 
3:    $H_{init} \leftarrow \emptyset$ ;  $T_{init} \leftarrow \emptyset$            ▷ initialise variables for initial rows of  $H, T$ 
4:    $S \leftarrow \emptyset$                                        ▷ initialise variable for  $sk(P^{\ll})$ 
5:   while  $P \neq \emptyset$  do
6:      $(x, y) \leftarrow$  point in  $P$  with least  $x$ -coordinate
7:      $y' \leftarrow$  least value in  $T_{init}$  greater than  $y$ ;  $+\infty$  if none exists           ▷ search
8:     if  $y' = +\infty$  then
9:       append  $x$  to  $H_{init}$ ; append  $y$  to  $T_{init}$            ▷ start new antichain
10:    else
11:      replace  $y'$  by  $y$  in  $T_{init}$ ; append  $(x, y')$  to  $S$            ▷ extend antichain
12:      remove  $(x, y)$  from  $P$ 
13:     $(H_+, T_+) \leftarrow$  RSK( $S$ )                               ▷ recursive call
14:     $H \leftarrow$  tableau with initial row  $H_{init}$  and remaining rows  $H_+$ 
15:     $T \leftarrow$  tableau with initial row  $T_{init}$  and remaining rows  $T_+$ 
16:    return  $(H, T)$                                            ▷  $rsk(P^{\ll}) = (H, T)$ 

```

The computation of the initial row in the standard RSK algorithm (before the recursive call in line 13 of Algorithm 1) is essentially identical to the classical algorithm for the LIS problem [13, 10, 6]. In line 7, the canonical antichain for each of the n points can be found by binary search, therefore the whole initial row is obtained in time $O(n \log n)$. In total, there are at most n rows in $rsk(P^{\ll})$, therefore the overall time is $n \cdot O(n \log n) = O(n^2 \log n)$.

Apart from the worst-case running time, it is of interest to consider the average-case running time of RSK algorithms on a uniformly random permutation; in this case, the shape of tableaux H, T turns out to be sampled from the *Plancherel* probability distribution (see, e.g. [17]). Romik [16] established this average-case running time to be $O(n^{3/2} \log n)$.

RSK with linear search. Paradoxically, a speedup can be obtained by replacing binary search with (a carefully controlled) linear search. Indeed, for a given x -coordinate, the value of the search target y' in line 7 of Algorithm 1 can only increase. Therefore, as long these different search invocations are performed as a linear search continuing from the search target of the previous invocation, the combined search time for a given x -coordinate will be $O(n)$, so the overall running time across all x -coordinates is reduced to $n \cdot O(n) = O(n^2)$. This observation may be considered part of the folklore; it is made e.g. by Thomas and Yong [21], who attribute it to an anonymous referee. A simple and elegant alternative description of this algorithm can be obtained by using edge local rules of Viennot [25], giving the same asymptotic running time $O(n^2)$.

Symmetric RSK algorithm. Felsner and Wernisch [9] proposed a more efficient, symmetric approach to developing an RSK algorithm. Their algorithm was described in the language of k -chains. In particular, they gave an algorithm for computing maximum k -chains (and, by symmetry, also k -antichains) of a planar point set in time $O(kn \log n)$. In combination with the algorithm for the same problem by Viennot [24], running in time $O((n^2/k) \log n)$, maximum k -chains can be obtained in time $O(n^{3/2} \log n)$ for all k .

Here, we give a simpler, more direct description of this combined algorithm of [24, 9], as an extension of the standard RSK algorithm. The main idea of the symmetric RSK algorithm is to construct the pair of tableaux $rsk(P^{\ll}) = (H, T)$ simultaneously by rows and by columns. The successive rows of tableaux H, T are constructed as in the standard RSK algorithm. At the same time, the successive columns in tableau H (respectively, T) are obtained by running the standard RSK algorithm for $rsk(P^{\leq})$ (respectively, $rsk(P^{\geq})$), using the symmetries exposed by Theorem 12.

There is clearly some redundancy in running the standard algorithm three times on partial orders $P^{\ll}, P^{\leq}, P^{\geq}$. However, this constant-factor redundancy allows one to reduce the overall asymptotic running time. Notice that as a result of the first iteration of each of the three runs, we obtain the union of the initial row and initial column in each of H and T ; this union is called the initial *principal hook* of the respective tableau. Likewise, as a result of the second iteration, we obtain the second principal hook of both H and T (i.e. the union of the second row and column, minus the initial principal hook). Crucially, while the number of both rows and columns in a Young tableau of order n can be as high as n , the number of its principal hooks is always at most $n^{1/2}$. Thus, the algorithm can be terminated after at most $\lfloor n^{1/2} \rfloor$ iterations made by each of the three simultaneous runs on $P^{\ll}, P^{\leq}, P^{\geq}$. The worst-case running time of the symmetric RSK algorithm is $n^{1/2} \cdot O(n \log n) = O(n^{3/2} \log n)$.

► **Example 16.** Figure 2 shows the execution of the symmetric RSK algorithm on the same input point set as in Figure 1. For the sake of brevity, only the computation of tableau T from partial orders P^{\ll}, P^{\geq} is shown explicitly, while the symmetric computation of tableau H from partial orders P^{\ll}, P^{\leq} is omitted. Compared to the three iterations of the standard algorithm in Figure 1, now only two iterations are required.

4 Speeding up RSK by doubling search

Doubling search. The *doubling search* technique (also called *exponential search*) was introduced by Bentley and Yao [1], and represents a hybrid between linear and binary search. Doubling search is particularly efficient for a non-uniform distribution of the target index, skewed towards an end of the array being searched.

We describe doubling search with the starting point at the upper end of the array, in order to be consistent with its intended application as a subroutine for RSK. Given an array $a_i, 1 \leq i \leq s$, sorted in increasing order, and a value q distinct from all a_i , we consider the problem of finding the greatest value in a less than q , that is index $k \geq 0$ such that $a_{s-k} < q < a_{s-k+1}$. We assume $a_i = -\infty$ for $i \leq 0$, and $a_{s+1} = +\infty$.

The search begins at the upper end of the array, comparing q against a_s . If $a_s < q$, we have found $k = 0$. Otherwise, the search continues in two phases. In the *doubling phase*, we compare q against $a_{s-1}, a_{s-2}, a_{s-4}, a_{s-8}, \dots$, until we find a subtrahend t that is the least power of 2 such that $a_{s-t} < q$. This phase takes $\lfloor \log k \rfloor + 1$ comparisons.

We now know that $1 \leq k \leq t$, and move on to the *binary search phase*. In this phase, we find the exact value of k in this range by binary search, taking at most $\lfloor \log t \rfloor \leq \lfloor \log k \rfloor + 1$ comparisons. Overall, the doubling search algorithm takes at most $2\lfloor \log k \rfloor + 3$ comparisons. Algorithm 2 shows the pseudocode for the doubling search algorithm.

Symmetric RSK with doubling search. Unfortunately, the asymptotic speedup by a factor of $n^{1/2}$ to the standard RSK algorithm, which is provided by the symmetric algorithm, is not compatible with the speedup by a factor of $\log n$ provided by linear search. However, we are still able to obtain both speedups simultaneously by employing doubling search.

■ **Algorithm 2** Doubling search.

```

1: procedure DSEARCH( $a, q$ )      ▷ given sorted array  $a$  and  $q$ , returns index for  $q$  in  $a$ 
2:   if  $a_s < q$  then return 0
3:    $t \leftarrow 1$ 
4:   while  $a_{s-t} > q$  do  $t \leftarrow 2t$                                 ▷ doubling phase
5:   return index  $k$  in  $\{1, \dots, t\}$ , such that  $a_{s-k} < q < a_{s-k+1}$     ▷ binary search

```

Consider a specific value x for a point's x -coordinate, as the RSK algorithm iterates on P , $sk(P^{\ll})$, $sk(sk(P^{\ll}))$, etc. These iterations form respectively row 1, 2, 3, ... of diagrams H, T . Let l_r denote the length of row r before a point with coordinate x is processed for that row. Let b_r denote the index of the search target y' in row r of a point with coordinate x (as per lines 9 or 11 of Algorithm 1); b_r is undefined if no point with coordinate x is left in iteration r (that is, in the $r - 1$ -th skeleton of P). Let the *displacement interval* in row r be $\{b_r, b_r + 1, \dots, \min(l_r, b_{r-1} - 1)\}$, where $b_0 = +\infty$. We denote this interval's length by $d_r = \min(l_r + 1, b_{r-1}) - b_r$; in particular, $d_r = 0$ if $b_r = b_{r-1}$. We also define $d_r = 0$ if b_r is undefined due to no point with coordinate x being left in iteration r .

► **Example 17.** Consider the computation of $rsk(P^{\ll})$ by the standard and the symmetric RSK algorithms in Figures 1 and 2. Let us fix $x = 6$.

In the first iteration, the bottom row of tableaux H, T is formed. Just before the processing of point $(6, 3) \in P$ begins, the current state of the tableaux rows is $H_{init} = (1, 2, 4)$, $T_{init} = (1, 5, 6)$, and their common length is $l_1 = 3$. The least value in T_{init} greater than $y = 3$ is 5, and its index in T_{init} is $b_1 = 2$. The displacement interval is between $b_1 = 2$ and $l_1 = 3$ inclusive, and its length is $d_1 = l_1 + 1 - b_1 = 2$.

In the second iteration, the middle row of tableaux H, T is formed. Just before the processing of point $(6, 5) \in sk(P^{\ll})$ begins, the current state of the tableaux rows is $H_{init} = (3, 5)$, $T_{init} = (2, 8)$, and their common length is $l_2 = 2$. The least value in T_{init} greater than $y = 5$ is 8, and its index in T_{init} is $b_2 = 2$. The displacement interval is empty (being defined between $b_2 = 2$ and $b_1 - 1 = 1$ inclusive), and its length is $d_2 = b_1 - b_2 = 0$.

In the third and final iteration (which is absent from the symmetric algorithm in Figure 2), the top row of tableaux H, T is formed. Just before the processing of point $(6, 8) \in sk(sk(P^{\ll}))$ begins, the tableaux rows H_{init}, T_{init} are both empty, and their common length is $l_3 = 0$. The least value in T_{init} greater than $y = 5$ is by convention $+\infty$, and its index in T_{init} is by convention $b_3 = 1$. The displacement interval is between $b_3 = 1$ and $b_2 = 2$, and its length is $d_3 = b_3 - b_2 = 1$.

► **Theorem 18.** *The symmetric RSK algorithm with doubling search solves the RSK correspondence problem in worst-case time $O(n^{3/2})$.*

Proof. Without loss of generality, assume that n is a perfect square (otherwise, the input can be extended by extra points with a suitably high y -value). Let $m = n^{1/2}$.

For a fixed x -coordinate, consider the displacement interval in a given row r . The rectangle of tableau cells below and including this interval consists of rd_r cells. All these rectangles for different values of r are pairwise disjoint. The symmetric RSK algorithm terminates after processing at most m rows. The total number of cells in the rectangles defined by the displacement intervals in these rows is obviously at most n :

$$\sum_{r=1}^m rd_r \leq n$$

We also have $\sum_{r=1}^m r = \frac{m(m+1)}{2} \leq m^2 = n$. Let $d'_r = d_r + 1$. By the above, we have

$$\sum_{r=1}^m r d'_r \leq n + n = 2n$$

While working on a point with coordinate x within row r , doubling search makes at most $\lfloor 2 \log d'_r \rfloor + 3$ comparisons. For the total number of comparisons made for the given x -coordinate, we have by the arithmetic-geometric mean inequality and the Stirling lower bound on the factorial (cancelling the rounding down of the logarithms, and omitting the constant factor 2 and the additive term $\sum_{r=1}^m 3 = O(m)$):

$$\begin{aligned} \sum_{r=1}^m \log d'_r &= \sum_{r=1}^m \log \frac{r d'_r}{r} = \log \prod_{r=1}^m \frac{r d'_r}{r} = \log \left(\frac{1}{m!} \prod_{r=1}^m r d'_r \right) \leq \\ &\log \left(\frac{1}{m!} \left(\frac{1}{m} \sum_{r=1}^m r d'_r \right)^m \right) \leq \log \frac{(2n/m)^m}{m!} = \log \frac{(2m)^m}{m!} \leq \log \frac{(2m)^m}{(m/e)^m} = \\ &m \log(2e) = O(m) \end{aligned}$$

There are n different x -coordinates to consider, therefore the algorithm makes $n \cdot O(m) = O(n^{3/2})$ comparisons in total. \blacktriangleleft

5 Conclusion

We have given a simple, direct description of the symmetric RSK algorithm by Felsner and Wernisch [9]. We have shown how this algorithm can be enhanced with doubling search, improving the asymptotic running time from $O(n^{3/2} \log n)$ to $O(n^{3/2})$. It is also worth noticing that the (worst-case) running time of our algorithm is lower than the average-case running time of the standard (or the symmetric) RSK algorithm on uniformly random permutations, as analysed by Romik [16]. Our result implies a similar improvement for the k -chain problem for arbitrary k .

A natural lower bound on the running time of RSK correspondence is provided by the LIS problem, which is a subproblem for RSK, and requires $\Omega(n \log n)$ comparisons in the comparison model [10]. Thus, there remains a substantial gap between the known upper and lower bounds for the asymptotic complexity of the RSK correspondence.

Apart from potential improvements in the algorithm or the lower bound, there is scope for future work in extending the algorithm for more general versions of the RSK correspondence, e.g. that between positive integer matrices and semistandard Young tableaux. An experimental confirmation of the efficiency of our algorithm also remains an endeavor for future work; this is a non-trivial task, since most existing experiments with RSK, e.g. those by Vasilyev and Duzhin [7, 22], concentrate on either Plancherel-random Young diagrams, or on Young diagrams with (near-)maximum dimensions; such a diagram shape seems to be far away from the worst-case shape suggested by the proof of Theorem 18.

References

- 1 Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–87, 1976. doi:10.1016/0020-0190(76)90071-5.
- 2 Sergei Bspamyatnikh and Michael Segal. Enumerating longest increasing subsequences and patience sorting. *Information Processing Letters*, 76:7–11, 2000. doi:10.1016/S0020-0190(00)00124-1.

- 3 Henrik Blunck and Jan Vahrenhold. In-Place Algorithms for Computing (Layers of) Maxima. *Algorithmica*, 57:1–21, 2010. doi:10.1007/s00453-008-9193-z.
- 4 Adam L. Buchsbaum and Michael T. Goodrich. Three-Dimensional Layers of Maxima. *Algorithmica*, 39:275–286, 2004. doi:10.1007/s00453-004-1082-5.
- 5 Jeff Calder, Selim Esedoğlu, and Alfred O. Hero. A PDE-based Approach to Nondominated Sorting. *SIAM Journal on Numerical Analysis*, 53:82–104, 2015. doi:10.1137/130940657.
- 6 E W Dijkstra. Some beautiful arguments using mathematical induction. *Acta Informatica*, 13(1):1–8, 1980. doi:10.1007/BF00288531.
- 7 V. S. Duzhin and N. N. Vasilyev. Asymptotic behavior of normalized dimensions of standard and strict Young diagrams: growth and oscillations. *Journal of Knot Theory and Its Ramifications*, 25(12):1642002, 2016. doi:10.1142/S0218216516420025.
- 8 P Erdős and G Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.
- 9 Stefan Felsner and Lorenz Wernisch. Maximum k -Chains in Planar Point Sets: Combinatorial Structure and Algorithms. *SIAM Journal on Computing*, 28:192–209, 1998. doi:10.1137/S0097539794266171.
- 10 Michael L Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11:29–35, 1975. doi:10.1016/0012-365X(75)90103-X.
- 11 Curtis Greene. An Extension of Schensted’s Theorem. *Advances in Mathematics*, 14:254–265, 1974.
- 12 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997. doi:10.1017/CB09780511574931.
- 13 D E Knuth. Permutations, matrices, and generalized Young tableaux. *Pacific Journal of Mathematics*, 34(3):709–727, 1970.
- 14 S. N. Majumdar and S. Nechaev. Exact asymptotic results for the Bernoulli matching model of sequence alignment. *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, 72(2):020901, 2005. doi:10.1103/PhysRevE.72.020901.
- 15 G de B Robinson. On the representations of the symmetric group. *American Journal of Mathematics*, 60:745–760, 1938. doi:10.2307/2372326.
- 16 D. Romik. The Number of Steps in the Robinson-Schensted Algorithm. *Functional Analysis and Its Applications*, 39(2):152–155, 2005. doi:10.1007/s10688-005-0030-8.
- 17 Dan Romik. *The Surprising Mathematics of Longest Increasing Subsequences*. Cambridge University Press, Cambridge, 2014. doi:10.1017/CB09781139872003.
- 18 C. Schensted. Longest Increasing and Decreasing Subsequences. *Canadian Journal of Mathematics*, 13:179–191, 1961. doi:10.4153/CJM-1961-015-3.
- 19 M. P. Schützenberger. Quelques remarques sur une construction de Schensted. *Mathematica Scandinavica*, 12:117–128, 1963. doi:10.7146/math.scand.a-10676.
- 20 Richard P. Stanley. *Algebraic Combinatorics*. Undergraduate Texts in Mathematics. Springer, New York, NY, 2013. doi:10.1007/978-1-4614-6998-8.
- 21 Hugh Thomas and Alexander Yong. Longest increasing subsequences, Plancherel-type measure and the Hecke insertion algorithm. *Advances in Applied Mathematics*, 46(1-4):610–642, 2011. doi:10.1016/j.aam.2009.07.005.
- 22 N. N. Vasiliev and V. S. Duzhin. A Study of the Growth of the Maximum and Typical Normalized Dimensions of Strict Young Diagrams. *Journal of Mathematical Sciences*, 216(1):53–64, 2016. doi:10.1007/s10958-016-2887-x.
- 23 G. Viennot. Une forme geometrique de la correspondance de Robinson-Schensted. In *Combinatoire et Représentation du Groupe Symétrique*, volume 579 of *Lecture Notes in Mathematics*, pages 29–58. Springer, 1977. doi:10.1007/BFb0090011.
- 24 G. Viennot. Chain and antichain families, grids and Young tableaux. *Annals of Discrete Mathematics*, 23:409–463, 1984. doi:10.1016/S0304-0208(08)73835-0.
- 25 X Viennot. Growth diagrams and edge local rules. In *Proceedings of GASCom*, 2018.

Insertion Time of Random Walk Cuckoo Hashing below the Peeling Threshold

Stefan Walzer  

Universität zu Köln, Germany

Abstract

Most hash tables have an insertion time of $\mathcal{O}(1)$, often qualified as “expected” and/or “amortised”. While insertions into cuckoo hash tables indeed seem to take $\mathcal{O}(1)$ expected time in practice, only polylogarithmic guarantees are proven in all but the simplest of practically relevant cases. Given the widespread use of cuckoo hashing to implement compact dictionaries and Bloom filter alternatives, closing this gap is an important open problem for theoreticians.

In this paper, we show that random walk insertions into cuckoo hash tables take $\mathcal{O}(1)$ expected amortised time when any number $k \geq 3$ of hash functions is used and the load factor is below the corresponding *peeling threshold* (e.g. ≈ 0.81 for $k = 3$). To our knowledge, this is the first meaningful guarantee for constant time insertion for cuckoo hashing that works for $k \in \{3, \dots, 9\}$.

In addition to being useful in its own right, we hope that our key-centred analysis method can be a stepping stone on the path to the true end goal: $\mathcal{O}(1)$ time insertions for all load factors below the *load threshold* (e.g. ≈ 0.91 for $k = 3$).

2012 ACM Subject Classification Theory of computation \rightarrow Bloom filters and hashing; Theory of computation \rightarrow Design and analysis of algorithms; Mathematics of computing \rightarrow Random graphs

Keywords and phrases Cuckoo Hashing, Random Walk, Random Hypergraph, Peeling, Cores

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.87

Related Version *Full Version:* <https://arxiv.org/abs/2202.05546>

Funding *Stefan Walzer:* DFG grant WA 5025/1-1.

Acknowledgements I would like to thank Martin Dietzfelbinger for providing several useful comments that helped with improving the presentation of this paper as well as an anonymous reviewer who gave useful feedback regarding the technical argument.

1 Introduction

Cuckoo Hashing Basics. Cuckoo hashing is an elegant approach for constructing compact and efficient dictionaries that has spawned both a rich landscape of theoretical results and popular practical applications. Briefly, each key x in a set of m keys is assigned k positions $h_1(x), \dots, h_k(x)$ in an array of $n \geq m$ buckets via hash functions h_1, \dots, h_k . Each bucket can hold at most ℓ keys and the challenge is to choose for each key one of its assigned buckets while respecting bucket capacities. We follow many previous works in assuming $k \geq 2$ and $\ell \geq 1$ to be constants and h_1, \dots, h_k to be uniformly random functions (but see e.g. [1, 2, 3] for works pursuing cuckoo hashing with explicit hash families).

Since the term cuckoo hashing was coined for $(k, \ell) = (2, 1)$ [27] and then generalised to $k \geq 3$ [12] and $\ell \geq 2$ [7], a major focus of theory papers has been to determine the load thresholds $c_{k,\ell}^*$, which are constants such that for a load factor $\frac{m}{n} < c_{k,\ell}^* - \varepsilon$ a placement of all keys exists with high probability (whp, defined in this paper as probability $1 - n^{-\Omega(1)}$) and for $\frac{m}{n} > c_{k,\ell}^* + \varepsilon$ a placement does not exist whp. This project has since been completed [11, 4, 6, 14, 13, 24] and we reproduce some thresholds in Table 1 for reference. Further research pursued cuckoo hashing variants with reduced failure probability [21], improved



© Stefan Walzer;

licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 87; pp. 87:1–87:11

Leibniz International Proceedings in Informatics



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

87:2 Insertion Time of Random Walk Cuckoo Hashing Below the Peeling Threshold

load thresholds [7, 23, 30] or weaker randomness requirements for h_1, \dots, h_k [1, 2, 3, 25, 22]. Moreover, cuckoo filters [9, 10, 8], which are Bloom filter alternatives based on cuckoo hashing, are now widely popular in the data base community.

■ **Table 1** For each $2 \leq k \leq 7$ and $1 \leq \ell \leq 6$ a cell shows $c_{k,\ell}^\Delta/\ell$ (left) and $c_{k,\ell}^*/\ell$ (right), rounded to three decimal places. The thresholds are divided by ℓ to reflect the corresponding memory efficiency (populated space over allocated space).

$\ell \setminus k$	2	3	4	5	6	7
1	– /0.500	0.818/0.918	0.772/0.977	0.702/0.992	0.637/0.997	0.582/0.999
2	0.838/0.897	0.776/0.988	0.667/0.998	0.579/1	0.511/1	0.457/1
3	0.858/0.959	0.725/0.997	0.604/1	0.515/1	0.450/1	0.399/1
4	0.850/0.980	0.687/0.999	0.562/1	0.476/1	0.412/1	0.364/1
5	0.837/0.990	0.658/1	0.533/1	0.448/1	0.387/1	0.341/1
6	0.823/0.994	0.635/1	0.511/1	0.427/1	0.368/1	0.323/1

Cuckoo Insertions. An important concern in all variants of cuckoo hashing is how to insert a new key x into an existing data structure. If all buckets $h_1(x), \dots, h_k(x)$ are full, then one key that is currently placed in those buckets has to be moved out of the way into one of its alternative buckets, which might require additional relocations of keys. Two natural strategies for organising insertions have been proposed [12]. Breadth-first search (BFS) insertion systematically pursues all possibilities for relocating keys in parallel. Random walk (RW) insertion starts by optimistically placing x into bucket $h_i(x)$ for a uniformly random $i \in [k]$ (where in this paper $[a] := \{1, \dots, a\}$ for $a \in \mathbb{N}$), without first considering any of the $k - 1$ other buckets. If $h_i(x)$ was already full, then a random key is evicted from $h_i(x)$ and is itself placed into one of its $k - 1$ alternative buckets. This chain of evictions continues until a bucket with leftover space is reached (see Figure 1 for the case with $\ell = 1$).

```

1 Algorithm RW( $x$ ):
2    $i_{\text{old}} \leftarrow \perp$ 
3   repeat
4     pick random  $i \in \{h_1(x), \dots, h_k(x)\} \setminus \{i_{\text{old}}\}$ 
5     swap( $x, B[i]$ )
6      $i_{\text{old}} \leftarrow i$ 
7   until  $x = \perp$ 

```

■ **Figure 1** The random walk insertion algorithm for $\ell = 1$. The array B of buckets is initialised with \perp .

Experiments suggest that, regardless of k and ℓ , and for any load factor $\frac{m}{n} < c_{k,\ell}^* - \varepsilon$ where insertions still succeed whp, the expected insertion time is independent of n , hence “ $\mathcal{O}(1)$ ” (neglecting dependence on the constants k, ℓ, ε), for both RW and BFS. Despite some partial success (see below), this claim has not been proven for any k and ℓ , neither for RW nor for BFS, with the exception of $(k, \ell) = (2, 1)$, which behaves very differently compared to other cases. A theoretical explanation for the good performance of cuckoo hashing in practical applications is therefore seriously lacking in this aspect. While this paper does not solve the problem, it puts a new kind of dent into it.

Contribution. Like most previous papers on cuckoo hashing insertions (an exception [18] is mentioned below) we focus on the case $\ell = 1$. Our analysis shows that RW insertions take $\mathcal{O}(1)$ expected amortised time for all $k \geq 3$, but it only works for $\frac{m}{n} < c_{k,1}^\Delta - \varepsilon$ where $c_{k,\ell}^\Delta < c_{k,\ell}^*$ is known as the *peeling threshold* or threshold for the occurrence of an $(\ell + 1)$ -core in a random k -uniform hypergraph [28, 26, 5, 19], see Table 1. Our analysis extends to a setting where the m insertions are carried out by m threads in parallel, each executing RW. We consider the worst case, where a (possibly adversarial) scheduler arbitrarily assigns the available computation time to threads that have not yet terminated. We only assume that the scheduler is oblivious of future random choices and that swaps are atomic, i.e. when several threads perform swaps concurrently, the effect is the same as executing these swaps in *some* sequential order (see e.g. [29, Sec. 2.4] for common parallel programming models).

- **Theorem 1.** *Let $k \in \mathbb{N}$ with $k \geq 3$ and $\varepsilon > 0$ be constants and $n, m \in \mathbb{N}$ with $\frac{m}{n} < c_{k,1}^\Delta - \varepsilon$.*
- (i) *Conditioned on a high probability event, sequentially inserting m keys into a cuckoo hash table with n buckets of size 1 using RW takes $\mathcal{O}(n)$ steps in expectation.*
 - (ii) *The same applies if the m insertions are started in parallel with arbitrary scheduling, only assuming that swaps are atomic. In other words, the combined work is $\mathcal{O}(n)$.*

Related Work and Comparison. Table 2 summarises related work that we now discuss from left to right.

[20] is only included here to show that the case of *static* cuckoo hash tables is well understood with optimal results in all considered categories.

[12] offers a strong analysis of BFS. The only downside is that it does not work for some small k and does not reach all the way to $c_{k,1}^*$. These issues might be resolvable by modernising the proof (the values $c_{k,1}^*$ were not known at the time of writing). Note, however, that even full success on this front would not render an analysis of RW irrelevant as several authors, including [12], see significant *practical* benefits of RW over BFS.

[17, 15] propose and improve, respectively, an analysis of RW via graph expansion. It guarantees most desired properties, including a concentration bound on insertion times. The major downside is an only polylogarithmic bound on expected insertion time.

[16] are first to prove an $\mathcal{O}(1)$ bound on expected random walk insertion time. The proof extends to any load factor $1 - \varepsilon$ with $\varepsilon > 0$. There is a downside, however. Instead of using $k = \mathcal{O}(\log(1/\varepsilon))$ hash functions as would be required for the existence of a placement of all keys (and as are used by [12] in their BFS analysis), the authors use $k = \mathcal{O}(\log(1/\varepsilon)/\varepsilon)$ hash functions. To give an example, while $k = 3$ hash functions suffice for $\varepsilon = 0.2$ (because $80\% < c_{3,1}^* \approx 92\%$), the analysis of [16] requires $k \geq 50$ hash

■ **Table 2** Guarantees offered by analyses on cuckoo table insertions. The motivation for the third line is that any load factor $< 50\%$ can be achieved with $k = 2$.

	[20]	[12]	[17, 15]	[16]	new
algorithm	offline construction	BFS	RW	RW'	RW
(expected amortised) insertion time	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\log^{\mathcal{O}(1)}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
least k for load factor $> 50\%$	3	≥ 10	3	≥ 12	3
supports load factor $1 - \varepsilon$ for large k	✓	✓	✓	✓	✗
supports load factor $c_{k,1}^* - \varepsilon$	✓	✗	✓	✗	✗
supports deletions	-	✓	✓	✗	✗

functions for that ε . Even if the analysis can be tuned to some degree (which seems probable), useful guarantees for practically relevant k would likely remain out of reach. We remark that they use a variant RW' of RW where a key x searches all its buckets $h_1(x), \dots, h_k(x)$ for a free space and only moves to a random bucket if all are full.

[18], not shown in the table, considers $k = 2$ hash functions, buckets of size $\ell \geq 2$ and random walk insertion. The result resembles [16] in its merits and downsides: Expected insertion time of $\mathcal{O}(1)$ is supported at any load factor $1 - \varepsilon$, provided that ℓ is large enough; the value $\ell = \ell(\varepsilon)$ required in the analysis is exponentially larger than what is needed for the existence of a placement; and meaningful guarantees for small values of ℓ seem out of reach.

This paper is the first to guarantee constant time insertions into cuckoo hash tables using $k \in \{3, \dots, 9\}$ hash functions. Like [16] our proof does not consider deletions. The main downside is that our analysis only works for $\frac{m}{n} < c_{k,1}^\Delta - \varepsilon$. Paradoxically, this means that the load factor supported by our analysis decreases when more hash functions are used (indeed, $c_{k,1}^\Delta \rightarrow 0$ for $k \rightarrow \infty$) and the supremum of supported load factors is $c_{3,1}^\Delta \approx 0.818$ for $k = 3$ hash functions. We mention a potential avenue for overcoming this problem in the conclusion.

Technical Overview. A central idea in our approach is to not count the number of evictions caused by a single insertion operation but to take the perspective of a single key and count how often it moves in the course of all m insertion operations combined.

Our proof is inspired by a simple observation: If a key x is assigned a bucket $h_i(x)$ that is assigned to no other key, then x is safely out of the way of other keys as soon as it has been placed in $h_i(x)$. In expectation, this happens after x has moved k times. A constant fraction of keys is “harmless” in this way. Moreover, there are keys y that are assigned a bucket $h_i(y)$ that is assigned to no other key, *except* for some harmless keys. It seems plausible that y , too, can quickly find a home in $h_i(y)$ and is only expected to be evicted from it a few times until the harmless keys live up to their name.

A formalisation attempt goes like this: Let F be an injective placement function assigning to each key x a bucket $F(x) \in \{h_1(x), \dots, h_k(x)\}$ (such an F exists whp for $c < c_{k,\ell}^* - \varepsilon$). We say a key x *depends* on a key y if $F(x) \in \{h_1(y), \dots, h_k(y)\}$, i.e. if the position designated for x is admissible for y . Let $D(x)$ be the set of all keys that x depends on. Finally, let $\text{moves}(x)$ be the total number of times that x moves during the insertion of all keys. We then have

$$\mathbb{E}[\text{moves}(x)] \leq k + \sum_{y \in D(x)} \mathbb{E}[\text{moves}(y)]. \quad (1)$$

The “ k ” is due to x being first placed in $F(x)$ after k moves in expectation. It can then only be evicted from $F(x)$ by a key from $D(x)$. Each movement of a key in $D(x)$ has a chance of $\frac{1}{k}$ to evict x from $F(x)$, causing k more moves of x in expectation until x is back in $F(x)$. Hence each move of a key from $D(x)$ can cause at most one move of x in expectation, as (1) suggests. The claim is even true in a more general context we call the *random eviction process* where in each round an adversary choses the key y to be moved among all keys not currently placed in their designated location $F(y)$.

As a way of bounding $\mathbb{E}[\text{moves}(x)]$, Equation (1) is hopelessly circular at first, but it is useful for specific F . Indeed, assume that the configuration of keys and buckets is *peelable*, i.e. for every subset X' of keys there is a bucket b^* assigned to only one key $x^* \in X'$. In that case, we can iteratively construct F , always picking such a pair (x^*, b^*) uniformly at random, setting $F(x^*) = b^*$ and removing x^* from further consideration. This yields an

acyclic dependence relation and an *acyclic dependence graph* (the directed graph with one vertex for each key that has the dependence relation as its edge relation). We can then upper bound $\mathbb{E}[\text{moves}(x)]$ by a multiple of $\text{peel}_F(x)$, which is the number of paths in the dependence graph that start at x . Bounding the expected number of moves of all insertion operations combined by $\mathcal{O}(n)$ then amounts to bounding the total number of paths in the dependence graph by $\mathcal{O}(n)$.

The second part of our argument – contained in the full version of this paper – is intimately related to the analysis of 2-cores in random hypergraphs. We extend Janson and Luczak’s “simple solution to the $(\ell + 1)$ -core problem” [19], which uses a random process embedded in continuous time where peeling is applied to the configuration model of a random hypergraph. We establish two guarantees concerning the peeling process.

- Firstly, the guarantee that during “early” rounds of peeling (when $\Omega(n)$ keys still remain) there are always $\Omega(n)$ candidate pairs (x^*, b^*) to choose from. Intuitively, this large number of choices for the peeling process makes it likely that the dependence graph becomes very “wide” with few long paths. For illustration (the formal argument works differently) assume the maximum path length is w with $w = \mathcal{O}(1)$. Since the indegree of the dependence graph is bounded by $k - 1$ this gives a bound of $m \cdot (k - 1)^w = \Theta(m)$ on the total number of paths as desired.
- Secondly, the technically demanding guarantee that in the “late” phase of peeling (when only $o(n)$ keys remain) almost all buckets have at most one remaining key assigned to them. Most steps of the peeling process will then not create further edges in the dependence graph. This implies that for each of the paths that already exist in the dependence graph less than one additional path is created in future rounds.

Outline. The rest of this paper is devoted to proving Theorem 1. We introduce two notable auxiliary concepts we call the *random eviction process* (REP) and *peeling numbers*. We reduce Theorem 1 to a claim about REP (Section 2) and reduce this claim to an upper bound on peeling numbers (Section 3). The remaining technical content is found in the full version of this paper. A deep dive into hypergraph peeling is required (full version Section 4). We then establish the upper bound on peeling numbers by counting paths in the dependence graph (full version Section 5).

2 Orientations, Peeling and the Random Eviction Process

In this section, we introduce the *random eviction process* (REP), which generalises sequential RW insertions, and formulate a claim on REP’ that implies Theorem 1.

From Hashing to Hypergraphs. A well-subscribed model for cuckoo hashing involves hypergraph terminology. The set of buckets corresponds to the set V of vertices and each key x corresponds to the hyperedge $\{h_1(x), \dots, h_k(x)\}$ in the set E of hyperedges. The task of placing all keys then becomes the task of *orienting* $H = (V, E)$ as explained below.

Under the *simple uniform hashing assumption*, the distribution of $H = H_{n,m,k}$ is simple: Each of the km incidences of the m hyperedges are chosen independently and uniformly at random from V . Formally this means that hyperedges are multisets of size k , possibly containing multiple copies of the same vertex (though in expectation only $\mathcal{O}(1)$ do) and E is a multiset possibly containing identical hyperedges (though whp E does not). This issue complicates a few definitions but does not cause any real trouble.

<pre> 1 Algorithm REP(V, E): 2 $f \leftarrow \{(e, \perp) \mid e \in E\}$ // i.e. $f \equiv \perp$ 3 while $\exists e \in E : f(e) = \perp$ do 4 pick such an e arbitrarily 5 pick a random $v \in e$ 6 if $\exists e' \neq e : f(e') = v$ then 7 $f(e') \leftarrow \perp$ 8 $f(e) \leftarrow v$ </pre>	<pre> 1 Algorithm REP'(V, E, F): 2 $f \leftarrow \{(e, \perp) \mid e \in E\}$ 3 while $\exists e \in E : f(e) \neq F(e)$ do 4 pick such an e arbitrarily 5 pick a random $v \in e$ 6 if $\exists e' \neq e : f(e') = v$ then 7 $f(e') \leftarrow \perp$ 8 $f(e) \leftarrow v$ </pre>
---	---

■ **Figure 2** The random eviction process (REP) is a generalisation of sequential random walk insertion. A variant REP' only terminates when a specific target orientation $F : E \rightarrow V$ is reached. Changes are highlighted in bold.

Orientations and Peelings. A *partial orientation* of a hypergraph $H = (V, E)$ is a function $f : E \rightarrow V \cup \{\perp\}$ with $f(e) \in e \cup \{\perp\}$ for each $e \in E$ that is injective except for collisions on \perp . If $f(e) = \perp$ then we say that e is *unoriented*, otherwise e is *oriented* (to $f(e)$). We call f an *orientation* if all $e \in E$ are oriented.

We can try to construct an orientation F of H greedily by repeatedly selecting a vertex v of degree 1 arbitrarily as long as one such vertex exists, setting $F(e) = v$ for the unique hyperedge e incident to v , and removing e from H . We call the resulting partial orientation F a *peeling* of H . If H does not contain a subhypergraph of minimum degree at least 2 (i.e. when the 2-core of H is empty [26]) then F is an orientation and we say H is *peelable*. We call F a *random peeling* if the choice of v is made uniformly at random whenever there are several vertices of degree 1.

The Random Eviction Process. The *random eviction process* (REP), see Figure 2 (left), is run on a hypergraph $H = (V, E)$ and maintains a partial orientation f of H . The process continues in a sequence of rounds as long as unoriented hyperedges remain, possibly indefinitely. In each round, an unoriented hyperedge e is chosen and oriented to a random incident vertex. If a different hyperedge e' was oriented to that vertex, then this e' is *evicted*, i.e. becomes unoriented.

A variant of REP is the *random eviction process with target orientation* (REP'), see Figure 2 (right). It is run on a hypergraph H and an orientation F of H . REP' works just like REP, except that it terminates only when $f = F$ is reached, and in every round any hyperedge e with $f(e) \neq F(e)$ may be chosen. We claim:

► **Proposition 2.** *Let $k \in \mathbb{N}$ with $k \geq 3$ and $\varepsilon > 0$ be constants and $n, m \in \mathbb{N}$ with $\frac{m}{n} < c_{k,1}^{\Delta} - \varepsilon$. Conditioned on a high probability event, $H = H_{n,m,k}$ is peelable and the random peeling F of H satisfies the following. REP' with target orientation F and an arbitrary¹ policy for choosing e in line 4 terminates after $\mathcal{O}(n)$ rounds in expectation.*

Proposition 2 is proved in the following section. We now show that it implies Theorem 1.

¹ This allows these choices to be made *adversarially*. The adversary may know all about H and the state of the algorithm but cannot predict future random choices made in line 5.

Proof of Theorem 1. The case of m sequential insertions is equivalent to the case of m parallel insertions where the scheduler only assigns computation time to the thread of least index that has not yet terminated. It therefore suffices to prove (ii), where the parallel case with *arbitrary* scheduling is considered.

We deal with m threads, each running RW, executed in an arbitrarily interleaved way. However, the only point where RW interacts with data visible to other threads is the swap, which is assumed to be atomic. A sufficiently general case is therefore one where the scheduler always picks an arbitrary thread that has not yet terminated and that thread is then allowed to run for one iteration of the loop. The correspondence between this process and REP should be clear: The scheduler's arbitrary choice of a thread implicitly chooses an unplaced key in that thread's local variable x , which is then placed into a random bucket, possibly evicting a different key. Likewise, REP arbitrarily chooses an unoriented hyperedge, which is then randomly oriented, possibly evicting another hyperedge.

For Proposition 2's claim on REP' to apply to RW, there are two differences to consider. **REP vs. REP'.** Assume an adversary wants to *maximise* the expected running times of REP and REP' by making bad choices for e in line 4. Her job is *harder* for REP for two reasons: Firstly, the termination condition is strictly weaker, such that REP may terminate when REP' does not. Secondly, her choices for e are restricted to unoriented hyperedges, where REP' additionally permits oriented hyperedges with $f(e) \neq F(e)$.

Intuitively speaking, the relatively weaker adversary in REP means that the upper bound on expected running time in Proposition 2 carries over from REP' to REP. More formally, every policy P for line 4 of REP is also valid for REP' and under the natural coupling random coupling REP' with P takes always at least as long as REP with P .

REP vs. RW: i_{old} . In RW an evicted key is not allowed to immediately move back into the bucket i_{old} it was just evicted from. The intuition is that this avoids a needless back-and-forth that otherwise occurs in 1 out of every k evictions. However, the author is not aware of a simple proof that the use of i_{old} is an improvement. Instead, we will check that the relevant part of the argument (Lemma 3) works for both cases. ◀

3 Bounding the Number of Evictions using Peeling Numbers

We now introduce the concept of peeling numbers and bound the number of evictions occurring in REP' in terms of them. This proves Proposition 2 but leaves the task of bounding peeling numbers for the full version of this paper.

Direct Dependence and Numbers of Moves. Consider a peelable hypergraph $H = (V, E)$ and a peeling $F : E \rightarrow V$ of H . For $e \neq e' \in E$ we say that e *directly depends* on e' if $F(e) \in e'$. This implies that e is peeled after e' , making the transitive closure of direct dependence an acyclic relation. We define $D(e) = D_F(e)$ as the set of all e' that e directly depends on, or more precisely: $D(e)$ is a multiset containing e' with the same multiplicity with which e' contains $F(e)$.

Now consider a run of REP' with target orientation F (and an arbitrary policy for line 4). For $e \in E$ let $\text{moves}(e)$ be the number of times that e is selected in line 4 of REP' (this is one more than the number of times that e is evicted).

► **Lemma 3.** For any $e \in E$ we have $\mathbb{E}[\text{moves}(e)] \leq k + \sum_{e' \in D(e)} \mathbb{E}[\text{moves}(e')]$.

Proof. For clarity, we ignore complications that are due to multisets at first. Let m_1 be the number of times that e moves until $f(e) = F(e)$ holds for the first time. Whenever e is selected to be moved, the chance to select $f(e) = F(e)$ is $\frac{1}{k}$, so clearly $\mathbb{E}[m_1] = k$. Afterwards,

e may not be selected anymore until evicted. Only hyperedges in $D(e)$ can evict e from $F(e)$ and when selected they do so with probability $\frac{1}{k}$, causing another k moves of e in expectation. It follows that $\mathbb{E}[m_+] = \mathbb{E}[m_D]$ where $m_+ := \text{moves}(e) - m_1$ and where m_D is the number of times that a hyperedge from $D(e)$ moves *while* $f(e) = F(e)$. The claim now follows from $m_D \leq \sum_{e' \in D(e)} \text{moves}(e')$ and linearity of expectation.

When $D(e)$ is a multiset the argument can be adapted: Whenever a hyperedge e' moves that is contained in $D(e)$ with multiplicity $a > 1$ it has an increased chance of $\frac{a}{k}$ to move to $F(e)$. But this is reflected in our bound since $\mathbb{E}[\text{moves}(e')]$ is counted a times.

Let us now consider a variant of the claim that incorporates the “ i_{old} ” feature of RW as promised in the proof of Theorem 1. In particular, a hyperedge never moves into the position it was last evicted from. We now have $\mathbb{E}[m_1] < k$ because all moves after the first move have an improved chance of $\frac{1}{k-1}$ to select $F(e)$. To compare $\mathbb{E}[m_+]$ and $\mathbb{E}[m_D]$, we can distinguish two kinds of moves. Concerning moves *away from* $F(e)$, m_D counts the same or one more compared to m_+ . All other moves have a chance of $\frac{1}{k-1}$ to end in $F(e)$ and contribute the same amount to $\mathbb{E}[m_+]$ and $\mathbb{E}[m_D]$ as before. The same adaptation to multisets applies. ◀

The Peeling Number. We define the peeling number of $e \in E$ recursively as

$$\text{peel}(e) = \text{peel}_F(e) := \sum_{e' \in D_F(e)} (1 + \text{peel}(e')). \quad (2)$$

Peeling numbers are well-defined by acyclicity of direct dependence, the base case being $\text{peel}(e) = 0$ for any e with $D(e) = \emptyset$. The idea is that $\text{peel}(e)$ counts the number of hyperedges that e directly *or indirectly* depends on, in other words, those hyperedges e' that must be peeled before e can be peeled. However, some e' may be counted multiple times. The relevance of peeling numbers lies in the following lemma.

► **Lemma 4.** *Let H be a peelable hypergraph with a peeling F . Let R be the number of rounds until REP' with target orientation F terminates. We have $\mathbb{E}[R] \leq k \cdot (m + \sum_{e \in E} \text{peel}_F(e))$.*

Proof. For a single $e \in E$ we have $\mathbb{E}[\text{moves}(e)] \leq k \cdot (1 + \text{peel}(e))$ because

$$\begin{aligned} \mathbb{E}[\text{moves}(e)] &\stackrel{\text{Lem.3}}{\leq} k + \sum_{e' \in D(e)} \mathbb{E}[\text{moves}(e')] \stackrel{\text{Induction}}{\leq} k + \sum_{e' \in D(e)} k \cdot (1 + \text{peel}(e')) \\ &\stackrel{\text{Eq.(2)}}{=} k + k \cdot \text{peel}(e) = k \cdot (1 + \text{peel}(e)). \end{aligned}$$

Since the total number R of rounds of REP' is the sum of all moves we conclude

$$\mathbb{E}[R] = \mathbb{E}\left[\sum_{e \in E} \text{moves}(e)\right] \leq \sum_{e \in E} k \cdot (1 + \text{peel}(e)) = k \cdot (m + \sum_{e \in E} \text{peel}(e)). \quad \blacktriangleleft$$

The remaining technical challenge is to bound the sum of all peeling numbers:

► **Proposition 5.** *Let H be as in Proposition 2. There is a high probability event \mathcal{E} such that, conditioned on \mathcal{E} , H is peelable and the peeling numbers with respect to the random peeling F of H satisfy*

$$\mathbb{E}\left[\sum_{e \in E} \text{peel}_F(e) \mid \mathcal{E}\right] = \mathcal{O}(n).$$

A prove is found in the full version of this paper (Section 5) and requires a detailed analysis of the peeling process (Section 4). We conclude this extended abstract with showing how Proposition 5 implies Proposition 2.

Proof of Proposition 2. We take the opportunity to clarify the structure of our probability space. There are three random experiments, performed in sequence: First, we pick a random hypergraph H . Second, if H is peelable, we pick a random peeling F of H and observe the peeling numbers. Last, we execute $\text{REP}'(H, F)$ and observe which moves are made. Note that the high probability event \mathcal{E} from Proposition 5 only relates to the first two steps (it does not relate to any moves). Lemma 4 only relates to the last step and does not require H and F to be random. For the number R of rounds of REP' we obtain:

$$\begin{aligned} \mathbb{E}[R \mid \mathcal{E}] &= \mathbb{E}[\mathbb{E}[R \mid H, F, \mathcal{E}] \mid \mathcal{E}] = \mathbb{E}[\mathbb{E}[R \mid H, F] \mid \mathcal{E}] \stackrel{\text{Lem. 4}}{\leq} \mathbb{E}\left[k \cdot (m + \sum_{e \in E} \text{peel}(e)) \mid \mathcal{E}\right] \\ &= km + k \cdot \mathbb{E}\left[\sum_{e \in E} \text{peel}(e) \mid \mathcal{E}\right] \stackrel{\text{Prop. 5}}{=} km + k \cdot \mathcal{O}(n) = \mathcal{O}(n). \quad \blacktriangleleft \end{aligned}$$

4 Conclusion and Future Work

This paper proves $\mathcal{O}(1)$ expected amortised running times for random walk insertions into cuckoo hash tables and is the first to yield meaningful results for small values of k such as $k = 3$. Our proof strategy is to link the number of times that a key x moves to the number of times that certain other keys move, where these other keys all precede x in the peeling process. The main technical challenge (addressed in the full version of this paper) was to extend an existing analysis of this peeling process in order to obtain stronger guarantees on its late stages when a sublinear number of keys remain. There are several ways in which our result might be strengthened.

- While amortisation is central to our argument, it seems unlikely to be required for the result itself. Indeed, the plausible claim that the expected time for inserting the i -th key is monotonically increasing in i already implies a non-amortised result.
- To make the result more relevant to practitioners, it is natural to pursue a generalisation to long sequences of insertions *and deletions* and to buckets of size $\ell \geq 2$. The author suspects that the given argument can be correspondingly extended with moderate technical complications.

If deletions are allowed then it seems natural to partition the data structure's lifetime into time slices of εn operations such that the set S_t of all keys present at some point during time slice t yields a peelable configuration whp. One would hope to conclude that $\mathcal{O}(n)$ evictions occur during the time slice in expectation whp. However, the peeling number of a key is no longer sufficient for bounding its expected number of moves for the simple reason that the key itself might be inserted and deleted frequently within the time slice.

- The most important goal, however, is to obtain a result that works up to the load threshold (for all $c < c_{k,1}^* - \varepsilon$), not just up to the peeling threshold (for $c < c_{k,1}^\Delta - \varepsilon$). There is at least one reason for optimism, namely the recent discovery of a variant of cuckoo hashing that raises the peeling threshold to the load threshold [31]. Briefly, a key's k hashes are randomly distributed in a random window of γn consecutive buckets. The peeling threshold of this variant is equal to $c_{k,\ell}^* - \varepsilon$ where $\varepsilon(\gamma)$ can be made arbitrarily small. However, when using this variant, an analysis can no longer rely on the configuration model due to a lack of symmetry between the vertices, meaning that even if the general idea is still sound, the proof would have to use different methods.

Regardless of whether such improvements are achievable, we believe this paper to be a promising step forward in the ongoing project of retrofitting the widespread use of cuckoo hash tables and cuckoo filters with strong theoretical guarantees.

References

- 1 Anders Aamand, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. Power of d choices with simple tabulation. In *45th ICALP*, volume 107 of *LIPICs*, pages 5:1–5:14, 2018. doi:10.4230/LIPICs.ICALP.2018.5.
- 2 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. doi:10.1007/s00453-013-9840-x.
- 3 Michael A. Bender, Tsvi Kopelowitz, William Kuzmaul, Ely Porat, and Clifford Stein. Incremental edge orientation in forests. In *29th ESA*, volume 204 of *LIPICs*, pages 12:1–12:18, 2021. doi:10.4230/LIPICs.ESA.2021.12.
- 4 Julie Anne Cain, Peter Sanders, and Nicholas C. Wormald. The random graph threshold for k -orientability and a fast algorithm for optimal multiple-choice allocation. In *Proc. 18th SODA*, pages 469–476, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283433>.
- 5 Colin Cooper. The cores of random hypergraphs with a given degree sequence. *Random Struct. Algorithms*, 25(4):353–375, 2004. doi:10.1002/rsa.20040.
- 6 Martin Dietzfelbinger, Andreas Goerdts, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight thresholds for cuckoo hashing via XORSAT. In *Proc. 37th ICALP (1)*, pages 213–225, 2010. doi:10.1007/978-3-642-14165-2_19.
- 7 Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.*, 380(1-2):47–68, 2007. doi:10.1016/j.tcs.2007.02.054.
- 8 David Eppstein. Cuckoo filter: Simplification and analysis. In *Proc. 15th SWAT*, pages 8:1–8:12, 2016. doi:10.4230/LIPICs.SWAT.2016.8.
- 9 Bin Fan, David G. Andersen, and Michael Kaminsky. Cuckoo filter: Better than Bloom. *login.*, 38(4), 2013. URL: <https://www.usenix.org/publications/login/august-2013-volume-38-number-4/cuckoo-filter-better-bloom>.
- 10 Bin Fan, David G. Andersen, Michael Kaminsky, and Michael Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proc. 10th CoNEXT*, pages 75–88, 2014. doi:10.1145/2674005.2674994.
- 11 Daniel Fernholz and Vijaya Ramachandran. The k -orientability thresholds for $g_{n,p}$. In *Proc. 18th SODA*, pages 459–468, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283432>.
- 12 Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst.*, 38(2):229–248, 2005. doi:10.1007/s00224-004-1195-x.
- 13 Nikolaos Fountoulakis, Megha Khosla, and Konstantinos Panagiotou. The multiple-orientability thresholds for random hypergraphs. *Combinatorics, Probability & Computing*, 25(6):870–908, 2016. doi:10.1017/S0963548315000334.
- 14 Nikolaos Fountoulakis and Konstantinos Panagiotou. Sharp load thresholds for cuckoo hashing. *Random Struct. Algorithms*, 41(3):306–333, 2012. doi:10.1002/rsa.20426.
- 15 Nikolaos Fountoulakis, Konstantinos Panagiotou, and Angelika Steger. On the insertion time of cuckoo hashing. *SIAM J. Comput.*, 42(6):2156–2181, 2013. doi:10.1137/100797503.
- 16 Alan M. Frieze and Tony Johansson. On the insertion time of random walk cuckoo hashing. *Random Struct. Algorithms*, 54(4):721–729, 2019. doi:10.1002/rsa.20808.
- 17 Alan M. Frieze, Páll Melsted, and Michael Mitzenmacher. An analysis of random-walk cuckoo hashing. *SIAM J. Comput.*, 40(2):291–308, 2011. doi:10.1137/090770928.
- 18 Alan M. Frieze and Samantha Petti. Balanced allocation through random walk. *Inf. Process. Lett.*, 131:39–43, 2018. doi:10.1016/j.ipl.2017.11.010.
- 19 Svante Janson and Malwina J. Luczak. A simple solution to the k -core problem. *Random Struct. Algorithms*, 30(1-2):50–62, 2007. doi:10.1002/rsa.20147.
- 20 Megha Khosla. Balls into bins made faster. In *Proc. 21st ESA*, pages 601–612, 2013. doi:10.1007/978-3-642-40450-4_51.

- 21 Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4):1543–1561, 2009. doi:10.1137/080728743.
- 22 Mathieu Leconte. Double hashing thresholds via local weak convergence. In *Proc. 51st Allerton*, pages 131–137, 2013. doi:10.1109/Allerton.2013.6736515.
- 23 Eric Lehman and Rina Panigrahy. 3.5-way cuckoo hashing for the price of 2-and-a-bit. In *Proc. 17th ESA*, pages 671–681, 2009. doi:10.1007/978-3-642-04128-0_60.
- 24 Marc Lelarge. A new approach to the orientation of random hypergraphs. In *Proc. 23rd SODA*, pages 251–264. SIAM, 2012. doi:10.1137/1.9781611973099.23.
- 25 Michael Mitzenmacher and Justin Thaler. Peeling arguments and double hashing. In *Proc. 50th Allerton*, pages 1118–1125, 2012. doi:10.1109/Allerton.2012.6483344.
- 26 Michael Molloy. Cores in random hypergraphs and Boolean formulas. *Random Struct. Algorithms*, 27(1):124–135, 2005. doi:10.1002/rsa.20061.
- 27 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. doi:10.1016/j.jalgor.2003.12.002.
- 28 Boris Pittel, Joel Spencer, and Nicholas C. Wormald. Sudden emergence of a giant k -core in a random graph. *J. Comb. Theory, Ser. B*, 67(1):111–151, 1996. doi:10.1006/jctb.1996.0036.
- 29 Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev. *Sequential and Parallel Algorithms and Data Structures - The Basic Toolbox*. Springer, 2019. doi:10.1007/978-3-030-25209-0.
- 30 Stefan Walzer. Load thresholds for cuckoo hashing with overlapping blocks. In *Proc. 45th ICALP*, pages 102:1–102:10, 2018. doi:10.4230/LIPIcs.ICALP.2018.102.
- 31 Stefan Walzer. Peeling close to the orientability threshold: Spatial coupling in hashing-based data structures. In *Proc. 32nd SODA*, pages 2194–2211. SIAM, 2021. doi:10.1137/1.9781611976465.131.

ParGeo: A Library for Parallel Computational Geometry

Yiqiu Wang ✉

CSAIL, MIT, Cambridge MA, USA

Rahul Yesantharao ✉

CSAIL, MIT, Cambridge MA, USA

Shangdi Yu ✉

CSAIL, MIT, Cambridge MA, USA

Laxman Dhulipala ✉

University of Maryland, College Park, MD, USA

Yan Gu ✉

University of California, Riverside, CA, USA

Julian Shun ✉

CSAIL, MIT, Cambridge, MA, USA

Abstract

This paper presents ParGeo, a multicore library for computational geometry. ParGeo contains modules for fundamental tasks including k d-tree based spatial search, spatial graph generation, and algorithms in computational geometry.

We focus on three new algorithmic contributions provided in the library. First, we present a new parallel convex hull algorithm based on a reservation technique to enable parallel modifications to the hull. We also provide the first parallel implementations of the randomized incremental convex hull algorithm as well as a divide-and-conquer convex hull algorithm in \mathbb{R}^3 . Second, for the smallest enclosing ball problem, we propose a new sampling-based algorithm to quickly reduce the size of the data set. We also provide the first parallel implementation of Welzl's classic algorithm for smallest enclosing ball. Third, we present the BDL-tree, a parallel batch-dynamic k d-tree that allows for efficient parallel updates and k -NN queries over dynamically changing point sets. BDL-trees consist of a log-structured set of k d-trees which can be used to efficiently insert, delete, and query batches of points in parallel.

On 36 cores with two-way hyper-threading, our fastest convex hull algorithm achieves up to 44.7x self-relative parallel speedup and up to 559x speedup against the best existing sequential implementation. Our smallest enclosing ball algorithm using our sampling-based algorithm achieves up to 27.1x self-relative parallel speedup and up to 178x speedup against the best existing sequential implementation. Our implementation of the BDL-tree achieves self-relative parallel speedup of up to 46.1x. Across all of the algorithms in ParGeo, we achieve self-relative parallel speedup of 8.1–46.61x.

2012 ACM Subject Classification Computing methodologies → Shared memory algorithms

Keywords and phrases Computational Geometry, Parallel Algorithms, Libraries

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.88

Related Version *Full Version:* <https://arxiv.org/abs/2207.01834>

Related Paper: <https://arxiv.org/abs/2112.06188>

Supplementary Material *Software (Source Code):* <https://github.com/ParAlg/ParGeo>

Funding This research is supported by DOE Early Career Award #DE-SC0018947, NSF CAREER Award #CCF-1845763, Google Faculty Research Award, Google Research Scholar Award, DARPA SDH Award #HR0011-18-3-0007, and Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA.



© Yiqiu Wang, Rahul Yesantharao, Shangdi Yu, Laxman Dhulipala, Yan Gu, and Julian Shun; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 88; pp. 88:1–88:19

Leibniz International Proceedings in Informatics



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

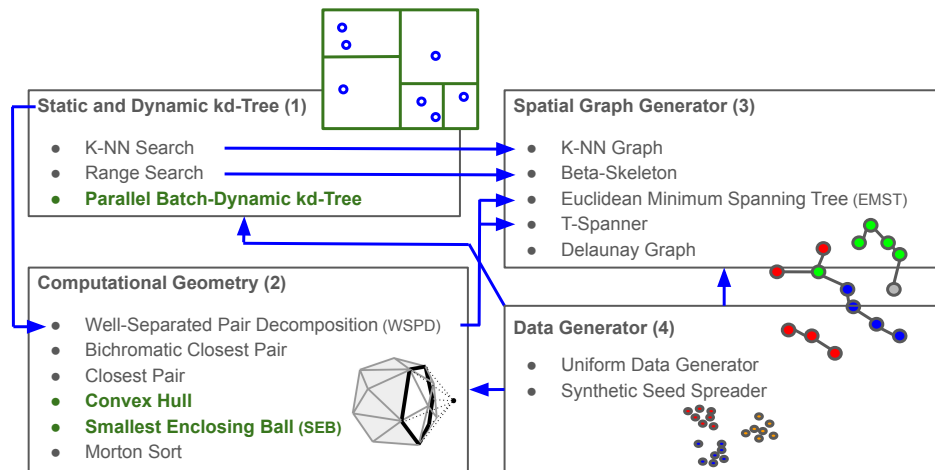
1 Introduction

Computational geometry algorithms have important applications in various domains, including computer graphics, robotics, computer vision, and geographic information systems [29, 43]. It is important to provide users with libraries of efficient computational geometry algorithms that they can easily use in their own higher-level applications. Furthermore, due to the growing sizes of data sets that need to be processed today, and the ubiquity of parallel (multicore) machines, it is beneficial to use parallel algorithms to speed up computations. In this paper, we present the ParGeo library for parallel computational geometry, which includes a rich set of parallel algorithms for geometric problems and data structures, including *kd*-trees, *k*-nearest neighbor search, range search, well-separated pair decomposition, Euclidean minimum spanning tree, spatial sorting, and geometric clustering. ParGeo also contains a collection of geometric graph generators, including *k*-nearest neighbor graphs and various spatial networks. Algorithms from ParGeo can either run sequentially, or run using parallel schedulers such as OpenMP, Cilk, or ParlayLib.

While there exist numerous libraries for computational geometry, most of them are not designed for parallel processing. For example, Libigl [35] is a library that specializes in the construction of discrete differential geometry operators and finite-element matrices. However, only some aspects of Libigl take advantage of parallelism. In comparison, the algorithms and implementations of ParGeo are designed for parallelism, and target a different set of problems. CGAL (Computational Geometry Algorithms Library) [2] is a well-known library of computational geometry algorithms that includes a wide range of algorithms, but most implementations are not parallel. Batista et al. [15] targeted a few important algorithms, including spatial sorting, box intersection, and Delaunay triangulation for shared-memory parallel processing, with code in CGAL. In comparison, ParGeo targets similar classes of problems as CGAL, but *all* of our implementations are highly parallel. PMP [47], Cinolib [39], and Tetwild [34] are libraries for polygonal and polyhedron meshes, tackling different problems from ParGeo. MatGeom [5] is a library for sequential geometric computing with MATLAB. The Problem Based Benchmark Suite [46, 12] is a multicore benchmark suite that has some overlap in algorithms with ParGeo. LEDA [40] is a library of data structures and algorithms for sequential combinatorial and geometric processing. ArborX [38] is a parallel library for spatial search.

In this paper, in addition to providing an overview of work on ParGeo, we describe new parallel algorithms implemented in ParGeo for convex hull, smallest enclosing ball, and batch-dynamic *kd*-tree that we developed. For convex hull, we develop new parallel algorithms for both \mathbb{R}^2 and \mathbb{R}^3 , where our key algorithmic novelty is a reservation technique to enable parallel modifications to the hull. For smallest enclosing ball, we propose a new sampling-based algorithm based on Larsson et al.'s [37] approach to quickly reduce the size of the data set. We also provide the first parallel implementation of the classic randomized incremental algorithm [27]. For *kd*-trees, we develop the BDL-tree, a new parallel data structure that supports batch-dynamic operations (construction, insertions, and deletions) as well as exact *k*-NN queries. BDL-trees consist of a set of exponentially growing *kd*-trees and perform batched updates in parallel.

To demonstrate the efficiency of our proposed algorithms and library, we perform a comprehensive set of experiments on synthetic and real-world geometric data sets, and compare the performance across our parallel implementations as well as optimized sequential baselines. On 36 cores with two-way hyper-threading, our best convex hull implementations achieve up to 44.7x (42.8x on average) self-relative speedup and up to 559x (325x on average) speedup against the best existing sequential implementation for \mathbb{R}^2 , and up to 24.9x (11.81x



■ **Figure 1** The figure shows an overview of modules in ParGeo. An arrow indicates that a component is used inside another component. In this paper, we present new algorithms and techniques for the modules highlighted in green.

on average) self-relative speedup and up to 124x (61.4x on average) speedup against the best existing sequential implementation for \mathbb{R}^3 . Our sampling-based smallest enclosing ball algorithm achieves up to 27.1x (20.08x on average) self-relative speedup and up to 178x (109x on average) speedup against the best existing sequential implementation for \mathbb{R}^2 and \mathbb{R}^3 . Our BDL-tree achieves self-relative speedup of up to 35.4x (30.0x on average) for construction, up to 35.0x (28.3x on average) for batch insertion, up to 33.1x (28.5x on average) for batch deletion, and up to 46.1x (40.0x on average) for full k -NN. Finally, across all implementations in ParGeo, we achieve self-relative parallel speedup of 8.1–46.6x (on average 23.2x).

2 The ParGeo Library

Our main goal in designing ParGeo was to enable reusable and efficient parallel implementations of geometric algorithms and data structures. We present an overview of the modules of ParGeo in Figure 1, highlighting how the modules interact with each other. ParGeo contains efficient multicore implementations of static and batch-dynamic kd -trees (Module (1)). The code supports kd -tree based spatial search, including k -nearest neighbor and range search. Our code is optimized for fast kd -tree construction by performing the split in parallel (either by spatial median or by object median), and performing the queries in a data-parallel fashion, which we will introduce in Section 5.

ParGeo contains a module for parallel computational geometry algorithms (Module (2)). Our kd -tree can be used to generate a well-separated pair decomposition [26] (WSPD), which can in turn be used to compute the hierarchical DBSCAN [52], ParGeo contains parallel implementations for the bichromatic closest pair, closest pair, convex hull, smallest enclosing ball, and Morton sorting.

In addition, ParGeo contains a collection of geometric graph generators (Module (3)) for point data sets. Our kd -tree’s k -NN search is used to generate the k -NN graph, and the range search is used to generate the β -skeleton graph [36]. Our WSPD generated from the kd -tree can also be used to compute the Euclidean minimum spanning tree [25, 52], and spanners [26]. ParGeo also generates the Delaunay graph.

■ **Table 1** Runtimes (seconds) and parallel speedups (T_1/T_{36h}) for PARGEO implementations on uniform hypercube data sets of varying dimensions and 10 million points. T_1 and T_{36h} denote the single-threaded and the 36-core hyper-threaded times, respectively. For batch-dynamic kd -tree updates, each batch contains 10% of the data set.

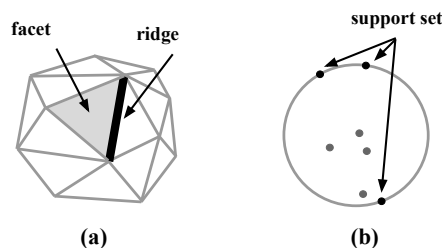
Implementation	T_1	T_{36h}	Speedup
<i>kd-tree Build (2d)</i>	5.51	0.43	12.70x
<i>kd-tree Build (5d)</i>	8.39	0.89	9.40x
<i>kd-tree k-NN (2d)</i>	31.45	0.68	46.34x
<i>kd-tree Range Search (2d)</i>	17.14	0.37	46.61x
<i>Batch-dynamic kd-tree Construction (5d)</i>	6.70	0.60	10.70x
<i>Batch-dynamic kd-tree Insert (5d)</i>	8.80	1.10	8.10x
<i>Batch-dynamic kd-tree Delete (5d)</i>	29.20	1.20	23.90x
<i>WSPD (2d)</i>	6.72	0.24	27.63x
<i>EMST (2d)</i>	33.02	1.58	20.86x
<i>Convex Hull (2d)</i>	0.38	0.0088	43.13x
<i>Convex Hull (3d)</i>	2.36	0.097	24.36x
<i>Smallest Enclosing Ball (2d)</i>	0.053	0.0033	16.30x
<i>Smallest Enclosing Ball (5d)</i>	0.13	0.014	9.54x
<i>Closest Pair (2d)</i>	10.35	0.52	19.90x
<i>Closest Pair (3d)</i>	28.00	2.32	12.07x
<i>k-NN Graph (2d)</i>	37.89	1.46	25.99x
<i>Delaunay Graph (2d)</i>	55.91	2.03	27.53x
<i>Gabriel Graph (2d)</i>	59.61	1.99	29.99x
<i>β-skeleton Graph (2d)</i>	113.27	3.20	35.37x
<i>Spanner (2d)</i>	27.19	2.15	12.67x

ParGeo contains a point data generator module (Module (4)) for which can generate uniformly distributed data sets, and clustered data sets of varying densities [31]. These data sets are used for benchmarking the other modules.

As shown in Table 1, on a machine with 36 cores with two-way hyper-threading, ParGeo achieves self-relative parallel speedups of 8.1–46.61x (23.15x on average) on a uniformly distributed data set, across all of the benchmarks. In the subsequent sections, we present three new algorithmic contributions provided in the library.

3 Convex Hull

The convex hull of a set of points P in \mathbb{R}^d is the smallest convex polyhedron containing P . It is common to represent the convex hull using a set of **facets**. The boundary of two facets is a **ridge**. For example, in \mathbb{R}^3 , assuming the points are in general position (no four points are on the same plane), each facet is a triangle, and each ridge is a line that borders two facets (see Figure 2(a)).



■ **Figure 2** (a) A facet and a ridge of a convex hull in \mathbb{R}^3 . (b) The support of the smallest enclosing ball in \mathbb{R}^2 .

The randomized incremental algorithm and the quickhull algorithm are the most widely used algorithms for solving convex hull in practice. The randomized incremental algorithm for \mathbb{R}^d was proposed by Clarkson and Shor [27]. Given a point data set P in \mathbb{R}^d , the randomized incremental algorithm first constructs a d -simplex, a generalization of a tetrahedron in d -dimensions as the initial hull. Then, the algorithm adds the points to the polyhedron in a random order, updating the hull if necessary. In practice, the quickhull algorithm [33, 14], another incremental algorithm, is often used. Unlike the randomized algorithm, the quickhull algorithm processes a point that is furthest from a facet, which enables the hull to be expanded more quickly. The quickhull algorithm is by far one of the most common implementations for convex hull due to its simplicity and efficiency [4, 6, 7, 1, 3, 2]. There have also been works that study parallel implementations of quickhull, but they are either limited to \mathbb{R}^2 [41, 48], or do not return the exact convex hull for \mathbb{R}^3 [49, 51]. Recently, Blleloch et al. [24] proposed a new randomized incremental algorithm that is highly parallel in theory. However, the algorithm does not seem to be practical due to numerous data structures required for bookkeeping.

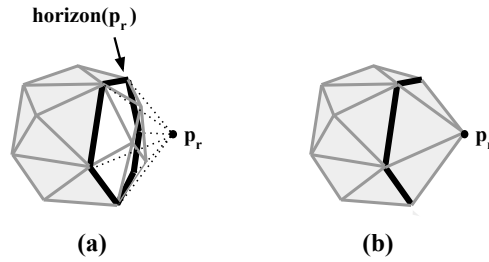
In this section, we describe our new parallel reservation-based algorithm. Our algorithm is able to express both the randomized incremental convex hull algorithm and the quickhull algorithm. Specifically, unlike a sequential incremental algorithm that adds one point per round, we add multiple points in parallel per round. We resolve conflicts caused by the parallel insertion using a reservation technique. We also apply a general parallelization technique based on divide-and-conquer, which in combination with our parallel incremental algorithm, leads to faster implementations in practice.

Parallel Reservation-Based Algorithm. Our parallel reservation-based algorithm can be implemented as either a randomized incremental algorithm or a quickhull algorithm. We will first introduce the overall structure of the algorithm. Then, we will describe the details about the implementations, and compare with existing approaches. We will base our description in the context of \mathbb{R}^3 for the sake of clarity, but the algorithm can be extended to \mathbb{R}^d for any constant integer $d \geq 2$.

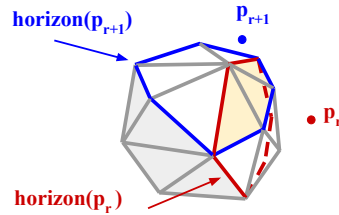
We first give a high-level overview of the algorithm. Given an ordered set of points $P = \{p_1, p_2, \dots, p_n\}$, we let $P_r = \{p_1, p_2, \dots, p_r\}$ be the prefix of P of size r , and $CH(P_r)$ be the convex hull on P_r . We start the construction by first arbitrarily selecting four points from P that do not lie on the same plane and putting them at the beginning of P , forming a tetrahedron $CH(P_4)$. Then, the algorithm proceeds iteratively, but on each round, rather than inserting just p_r to form $CH(P_r)$, we process a batch of points in parallel. On each round, let each point outside of $CH(P_{r-1})$ be called a **visible point**. We first select a batch of visible points, and try to add them to $CH(P_{r-1})$ in parallel in the same round.

The key challenge of this approach is that some of these points cannot be processed in parallel due to concurrent modifications on the shared structures of the convex polyhedron. We use a reservation algorithm to resolve these conflicts, such that we only process the points that modify disjoint facets of the polyhedron. Specifically, each point will perform a priority write [45] with its ID to reserve all of its visible facets. Points that have their ID written to all of its visible facets are *successful*. We then process the successful points in parallel by enabling them to make modifications to $CH(P_{r-1})$. At the end of the round, in parallel, we filter out points that are no longer visible. The algorithm will terminate when there are no more visible points.

We now describe the algorithm in more detail. Figure 3 illustrates the processing of a visible point p_r . We denote a facet as a **visible facet** of p_r if point p_r is in the half space away from the center of the convex hull. We first retrieve the set of visible facets of p_r via



■ **Figure 3** Illustration of adding a visible point p_r to the convex hull. (a) shows the convex hull prior to the addition of p_r . The visible facets are in white, while the non-visible facets are in gray. The thicker line segments correspond to the horizon. (b) shows the convex hull after adding p_r with newly created facets.



■ **Figure 4** This figure illustrates the attempt to add p_r and p_{r+1} in parallel. The visible points and horizons of p_r and p_{r+1} are in red and blue, respectively. The visible facets to either visible points are in white/yellow, while the other facets are in gray. The overlap of the three visible facets between the p_r and p_{r+1} is in yellow.

facets stored in it. The visible facets of p_r form a closed region, whose boundary is a set of ridges, known as the **horizon**. We delete the visible facets from $CH(P_{r-1})$, and replace them with new facets, where each new facet is formed by adding two ridges from a horizon ridge to p_r .

Because of the structural changes to the convex hull that occur when adding a visible point, concurrent structural changes can cause data races, which need to be avoided. We show an example of the conflict in Figure 4, where we are attempting to add two visible points p_r and p_{r+1} in parallel. As shown in the figure, the closed regions formed by the visible facets of each visible point overlap with each other in three facets, which are highlighted in yellow. Should the two visible points be processed in parallel, the resulting polyhedron may not be well-defined due to data races. When processed sequentially, p_{r+1} 's visible facets would have been different, involving newly created facets by p_r .

Our reservation algorithm allows only a subset of the visible points that update disjoint facets of the convex hull to be processed in parallel on each round. At a high level, we use the lexicographical order of the visible points to determine the priority in processing a facet (a smaller ID has higher priority). In the example shown in Figure 4, since p_r has a smaller ID than p_{r+1} , the three conflicting facets can only be processed by p_r in that round. The pseudocode for the algorithm is shown in Figure 5. P is processed iteratively until it is empty (Line 4). We allocate an extra data field in each facet for performing reservations (Lines 6–8). For each visible point in parallel, we iterate through its visible facets and use a parallel priority write (**WriteMin**) to write its ID to the facets' "reservation" fields. Then on Lines 9–11, we determine which visible points successfully reserved all of its facets. Again, in


```

1  Input: 3-dimensional points P, batch size r
2  Output: 3-dimensional convex hull
3  CH := initialize with 4 points
4  while (P is not empty):
5      Q := a batch of size r of visible points in P
6      par_for (q in Q): /* reservation */
7          for (f in q.visibleFacets):
8              WriteMin(&f.reservation, q.id)
9      par_for (q in Q): /* check reservation */
10         for (f in q.visibleFacets):
11             q.success &&= (f.reservation == q.id)
12     par_for (q in Q): /* process successful points */
13         if (q.success):
14             delete q's visible facets
15             create new facets of q
16             update CH
17     P := ParallelPack(P, visible)

```

■ **Figure 5** Pseudocode for the parallel reservation-based convex hull algorithm (which includes the randomized incremental algorithm and the quickhull algorithm).

parallel for each visible point, we check each of its visible facets for a successful reservation by comparing the value of the reservation field with its token. The reservation of a visible point is only successful if its ID is stored in all of its visible facets. Then, on Lines 12–16, we process the visible points whose reservations are successful, adding them to the hull and updating the appropriate data structures. Finally, on Line 17, we process the points in P such that those remaining as visible points are packed to replace the original P , and the non-visible points are discarded. Note that the visible points that succeeded in the reservation are no longer visible points because they are now part of the convex hull. Some of the remaining points will also no longer be visible points due to the growth of the convex hull.

We use a simple and fast data structure to keep track of the visibility relationship between the visible points and the facets. At each step of the algorithm, when a visible point is processed, it needs to identify the set of visible facets. On the other hand, for the facets undergoing structural changes, they need to identify and redistribute their visible points to new facets. To find the set of visible facets of p_r , it is inefficient to iterate through all of the facets of $CH(P_{r-1})$. While existing approaches [29] keep track of the visibility between visible points and *all* of their visible facets, we found such an approach to be slow because each vertex is associated with multiple facets, making the cost of storing and updating the data structure high. We only store the reference of an arbitrary visible facet to each visible point, from which we use a local breadth-first search to retrieve all of the visible facets only when needed. For storing the visible points in the facets, we assign each point to one of its visible facets. During point redistribution, we gather the points stored in each visible facet into an array, and in parallel distribute each point to a new visible facet replacing the original visible facet. Each such point also stores a reference to this visible facet.

Our reservation-based algorithm can be used to implement the parallel randomized incremental algorithm or the quickhull algorithm for convex hull. For the randomized incremental algorithm, we randomly permute the input points at the beginning, and on each round attempt to add a prefix of the permuted points to the convex hull. For the quickhull algorithm, on each round, we instead select a set of points furthest from a subset of facets.

We describe the implementation of the two algorithms in greater detail in the full version of our paper. Our reservation-based algorithm is inspired by the idea of “deterministic reservations” from Blleloch et al. [22], who introduce this approach to implement parallel algorithms for other problems. In the full version of our paper, we show the work overhead of doing reservations compared to the sequential algorithm is small.

Parallel Divide-and-Conquer. We adopt a common parallelization strategy using divide-and-conquer, which calls our reservation-based algorithm as a subroutine. Some early convex hull algorithms are based on divide-and-conquer, notably, the algorithm by Preparata and Hong [42]. The algorithm splits the input into two spatially disjoint subsets by a mid-point along one of the axis, recursively computes the convex hull on each subset, and then merges the results together. Later work [10, 28, 11] extended this approach to the parallel setting. However, most of these approaches rely on complicated subroutines to merge convex hulls, which are not practical and have not been implemented, to the best of our knowledge.

We implement a practical divide-and-conquer algorithm by partitioning the input into $c \cdot numProc$ equal subsets, where c is a small constant and $numProc$ is the number of processors. For each subset, the convex hull of the subset is computed by a single processor using the sequential quickhull algorithm, but run in parallel across the different subsets. Then, the vertices of the outputs of the subproblems are collected to form a new input, from which the final convex hull is computed using our reservation-based parallel algorithm described earlier.

Point Culling via Pseudohull Computation. We also implement a multicore variant of Tang et al.’s pseudohull heuristic [50], originally proposed for the GPU. Starting from an initial tetrahedra, we recursively grow each facet into three new facets, using the furthest point from the facet, similar to the quickhull algorithm. The visible points associated with the facet are redistributed to the new facets. This results in a polyhedron, and the points in the interior of the polyhedron will not be part of the convex hull. Therefore, we can prune away the points inside the polyhedron and compute the convex hull on the rest of the points.

There are several differences in our implementation from Tang et al.’s algorithm. Our implementation executes the recursive calls on different facets asynchronously in parallel, whereas Tang et al.’s implementation maps the algorithm to the GPU architecture by pre-allocating space for the facets and visible points, and runs the algorithm in an iterative manner in lock-step. Specifically, successively generated facets and points associated with them are updated by multiple threads in parallel in each iteration. We use a parallel maximum-finding routine to find the furthest point of each facet in each call. Rather than growing the pseudohull until there are no more visible points as done by Tang et al., we set a threshold on the number of points associated with a facet, below which we stop growing the pseudohull. This prevents stack overflow on large and skewed data sets due to too many recursive calls, and the extra unpruned points do not contribute significantly to the work of the final computation of the convex hull. At the end of pruning, we use our parallel reservation-based quickhull algorithm to compute the final hull on the remaining points, whereas Tang et al. uses a sequential implementation.

4 Smallest Enclosing Ball

The smallest enclosing ball of P in \mathbb{R}^d is the smallest d -sphere containing P . It is well known that the smallest enclosing ball is unique and defined by a **support set** of $d + 1$ points on the surface of the ball (see Figure 2(b)).

Welzl [53] showed that by using a randomized incremental algorithm, the smallest enclosing ball can be computed in $O(n)$ time in expectation for constant d . The algorithm iteratively expands the support set of the ball by adding points in a random order until the ball contains all of the points. The algorithm was later improved by Gartner [32] with practical optimizations for speed and robustness. Larsson et al. [37] proposed practical parallel algorithms that use a new method for expanding the support set, and their implementations work on both CPUs and GPUs. Later, Blleloch et al. [23] proposed a parallel algorithm based on Welzl’s algorithm, but without any implementations.

In this section, we describe our new algorithms for the smallest enclosing ball problem based on Larsson et al.’s approach [37]. We propose a sampling-based algorithm to quickly reduce the size of the data set. We also provide the first parallel implementation of Welzl’s classic algorithm.

Given a ball B , we define **visible points** to be points that lie outside of B . Existing approaches for computing the smallest enclosing ball focus on expanding the support set in an iterative manner, and output the enclosing ball when there are no more visible points. Welzl’s algorithm expands the support set by adding points in a random order [53]. In comparison, Larsson et al.’s approach scans the input to search for good support sets in a round-based manner. In \mathbb{R}^3 , Larsson’s algorithm divides the space into eight orthants centered at the center of B . On each round, the input is scanned to find the furthest visible points in each orthant. B is then updated to the next intermediate solution using the existing support set of B and the new visible points found during the scan. The algorithm iterates until there are no more visible points. It is parallelized within each round by performing the scan on the input in parallel.

Sampling-Based Algorithm. We find each iteration in Larsson et al.’s algorithm to be unnecessarily expensive due to having to scan the entire data set on every round. Our approach is to use a sampling heuristic to first obtain a good initial ball, inspired by Welzl’s randomized algorithm. Specifically, we use small random samples to obtain good estimates of the support set at a negligible cost.

We show the pseudocode of our algorithm in Figure 6. Our sampling-based algorithm consists of two phases: the sampling phase (Line 5–13) and the final compute phase (Line 15–20). First, we initialize the ball using a few arbitrary points (Line 3). Then, we iterate through a random permutation of the input to take multiple samples (Line 5–13). On each iteration, we scan through a constant-sized segment of the unseen part of the input, which is equivalent to a random sample. We perform an orthant scan similar to Larsson’s approach. Our implementation of orthant scan will return a new estimate of the support set based on the sample, and a boolean *hasOutlier* indicating whether the sample contains visible points with respect to the current smallest enclosing ball B (Line 7). We recompute B using the new support set. If there are visible points in the current sample, we will continue the sampling process with our new B . If there are no visible points in the sample, the support set likely contains most of the points, and so we terminate sampling and move on to the next phase. Now, with a good estimate of the optimal smallest enclosing ball, we run Larsson’s orthant scan to compute the final smallest enclosing ball (Line 15–20). The sampling phase allows us to generate good support sets without having to scan the entire input.

We parallelize the orthant scan, which is the most expensive operation of the algorithm. Specifically, we divide the input array to orthant scan into blocks, and process each block sequentially, but in parallel across different blocks. Afterward, the extrema for the orthants obtained from the blocks are merged, and a new support set is computed on these points and the existing support set of B .

```

1 Input: d-dimensional points P, batch size c
2 Output: d-dimensional smallest enclosing ball
3 B = ball()
4 /* Sampling phase */
5 scanned = 0
6 while (scanned < n):
7   hasOutlier, support =
8     orthantScan(P[scanned:min(scanned+c,n)-1],B)
9   scanned += c
10  if (!hasOutlier):
11    break /* current sample does not violate B */
12  else
13    B = constructBall(support)
14 /* Final computation phase */
15 while (hasOutlier):
16   hasOutlier, support = orthantScan(P, B)
17   if (!hasOutlier):
18     return B
19   else
20     B = constructBall(support)

```

■ **Figure 6** Pseudocode for the parallel sampling-based algorithm for smallest enclosing ball.

We parallelize the orthant scan, which is the most expensive operation of the algorithm. Specifically, we divide the input array to orthant scan into blocks, and process each block sequentially, but in parallel across different blocks. Afterward, the extrema for the orthants obtained from the blocks are merged, and a new support set is computed on these points and the existing support set of B .

Parallel Welzl's Algorithm and Optimizations. We also implemented and optimized the parallel version of Welzl's algorithm described by Blelloch et al. [23]. Welzl's sequential algorithm uses a random permutation of the input P and processes the points one by one. If the algorithm encounters a visible point p_i with respect to the current bounding ball B , B is recomputed on P_i , the prefix of points up until p_i , using recursive calls to the algorithm. Blelloch et al.'s parallel algorithm also uses a random permutation of P . Across iterations, the algorithm processes prefixes of P of exponentially increasing size. If the prefix contains at least one visible point, the earliest visible point p_i is identified, and B is recomputed on prefix P_i by recursively calling the parallel algorithm. Each prefix is processed in parallel.

We implement the algorithm with some practical optimizations. When there are numerous visible points in the prefix, the work of the parallel algorithm will increase significantly, because each time a visible point is discovered, the points after the visible point in the same prefix will have to be reprocessed in the next round. Therefore, given that there will be more visible points in the initial rounds when the prefix size is small (< 500000), we process these prefixes sequentially by calling Welzl's sequential algorithm. This also reduces the amount of overhead from parallel primitives, since there is limited parallelism for small prefixes.

In addition, we extend existing optimizations of Welzl's sequential algorithm to the parallel setting. We implement the move-to-front heuristic [53], which upon encountering a visible point, moves the visible point to the front of P , so that it will be processed earlier in recursive calls, reducing the number of subsequent visible points. We also parallelize the pivoting heuristic proposed by Gartner [32]. In this heuristic, upon encountering a visible

point, rather than processing the visible point directly, we search P for a *pivot point* furthest away from the center of the current B , and use the pivot point to compute the new B instead of the visible point. We use a parallel maximum-finding algorithm to identify the pivot point.

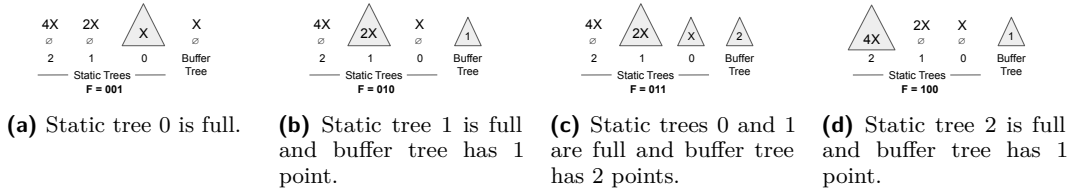
5 Parallel Batch-Dynamic k d-tree

The k d-tree, first proposed by Bentley [16], is a binary tree data structure that arranges and holds spatial data to speed up spatial queries. At each node, the data set is split into two using an axis-aligned hyperplane along a dimension, until the node holds a small constant number of points. k d-trees are used in a wide range of applications, such as in databases, machine learning, data compression, and cluster analysis.

In this section, we introduce the BDL-tree, a parallel batch-dynamic k d-tree implemented using the logarithmic method [17, 18]. Our BDL-trees build on ideas from the Bkd-Tree by Procopiuc et al. [44] and the cache-oblivious k d-tree by Agarwal et al. [9]. The logarithmic method [17, 18] for converting static data structures into dynamic ones is a very general idea. At a high level, the idea is to partition the static data structure into multiple structures with exponentially growing sizes (powers of 2). Then, inserts are performed by only rebuilding the smallest structure necessary to account for the new points. In the specific case of the k d-tree, a set of N_s static k d-trees is allocated, with capacities $[2^0, 2^1, \dots, 2^{N_s-1}]$, as well as an extra buffer tree with size 2^0 . Then, when an insert is performed, the insert cascades up from the buffer tree, rebuilding into the first empty tree with all the points from the lower trees. If desired, the sizes of all of the trees can be multiplied by a buffer size X , which is a constant that is tuned for performance.

We implement the underlying static k d-trees in an BDL-tree using the van Emde Boas (vEB) [13, 30, 9] recursive layout. Agarwal et al. [9] show that this memory layout can be used with k d-trees to make traversal cache-oblivious. We provide more details of the static tree structure, and parallel algorithms for the construction, deletion, and k -NN search in the full version of our paper.

Parallel Batch Insertion. Batch insertions are performed in the style of the logarithmic method [17, 18], with the goal of maintaining the minimum number of full trees within BDL-tree. Thus, upon inserting a batch P of points, we rebuild larger trees if it is possible using the existing points and the newly inserted batch. We use a bitmask to determine which static trees in the structure to destroy and reconstruct after each insertion. Specifically, we build a bitmask F of the current set of full static trees. Given the buffer k d-tree size X , we add the value $\lfloor |P|/X \rfloor$ to F when a point set P is inserted, after which the bitwise difference with the previous F indicates which trees need to be changed. We gather the points in the trees to be destroyed, and with P , we construct a subset of new trees in parallel. As an implementation detail, note that we first add $|P| \bmod X$ points to the buffer k d-tree – if we fill up the buffer k d-tree, then we gather the X points from it and treat them as part of P , effectively increasing the size of P by X . Refer to Figure 7 for an example of this insertion method ($X > 2$ in this example). In Figure 7a, the BDL-tree contains X points, giving a bitmask of $F = 1$ (because only the smallest tree is in use). If we insert $X + 1$ points, then we put one point in the buffer tree and compute $F_{new} = 1 + \lfloor \frac{X}{X} \rfloor = 2$, and so we have to deconstruct static tree 0 and build static tree 1, as shown in Figure 7b. Then, if we insert $X + 1$ points again, then we again put one point in the buffer tree and compute $F_{new} = 2 + \lfloor \frac{X}{X} \rfloor = 3$, and so we simply construct tree 0 on the X new points (leaving tree 1 intact), as seen in Figure 7c. Finally, if we then insert $X - 1$ points, this would fill the buffer



■ **Figure 7** A BDL-tree in various configurations with $X > 2$; starting from (a), inserting $X + 1$ points gives (b), then inserting $X + 1$ points gives (c), and then inserting $X - 1$ points gives (d).

up, and so we take 1 point from the buffer and insert X points; then, $F_{new} = 3 + \lfloor \frac{X}{X} \rfloor = 4$, and so we deconstruct trees 0 and 1, and construct tree 2, as seen in Figure 7d. We include a more detailed explanation of the algorithm, and explain the batch deletion algorithm in the full version of our paper.

Data-Parallel k -NN. In the data-parallel k -NN implementation, we parallelize over S , the set of points to search for nearest neighbors for. First, we allocate a k -NN buffer for each of the points in S . Then, iterating over each of the non-empty trees in the BDL-tree sequentially, we call the data-parallel k -NN subroutine on the tree, passing in the set S of points and the k -NN buffers. Because we reuse the same set of k -NN buffers for each k -NN call (note that each k -NN call is internally parallel), we end up with the k -nearest neighbors of the entire pointset for each point in S . We include a more detailed explanation in the full version of our paper.

6 Experimental Evaluation

Data Sets. We use several types of synthetic data sets. The first is **Uniform (U)**, consisting of points distributed uniformly at random inside a hypercube with side length \sqrt{n} , where n is the number of points. The second type **InSphere (IS)** is similar to the first, but the points are distributed in a hypersphere instead. We also use **OnSphere (OS)** and **OnCube (OC)** data sets, where points are uniformly distributed on the surface of a hypersphere and a hypercube, respectively. The surfaces have a thickness equal to 0.1 times the diameter or side length of the sphere or cube. We name the data sets in the format of **Dimension-Name-Size**.

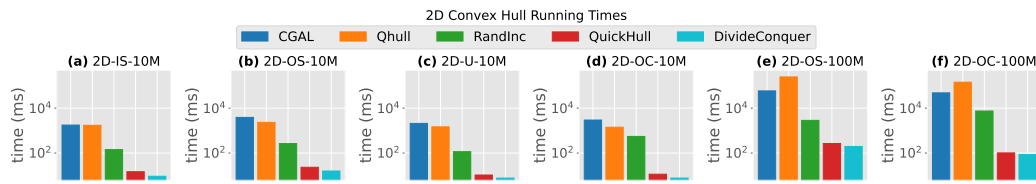
We also use the following real-world data sets from the Stanford 3D Scanning Repository [8]: **3D-Thai-5M** is a 3-dimensional point data set of size 4999996 from a scanned thai-stature; and **3D-Dragon-3.6M** is a 3-dimensional point data set of size 3609600 from a scanned statue of a dragon.

Testing Environment. The experiments are run on an AWS c5.18xlarge instance with 2 Intel Xeon Platinum 8124M CPUs (3.00 GHz), for a total of 36 two-way hyper-threaded cores and 144 GB RAM. We compile our benchmarks with the g++ compiler (version 9.3.0) with the `-O3` flag, and use ParlayLib [20] for parallelism.

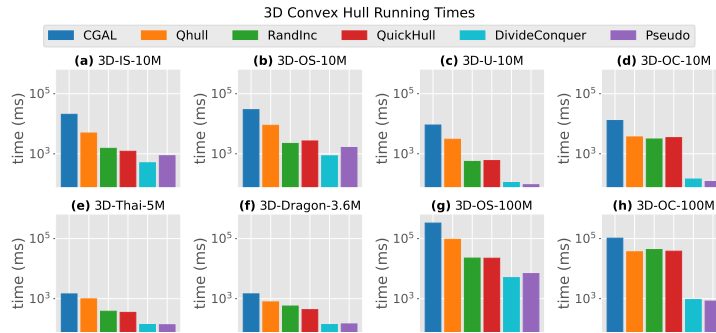
6.1 Convex Hull

We test the following implementations for convex hull (our new implementations are underlined). All implementations are for both \mathbb{R}^2 and \mathbb{R}^3 .

- **CGAL**: sequential C++ implementation of quickhull in CGAL [2].
- **Qhull**: sequential C++ implementation of quickhull [6] by Barber et al. [14].



■ **Figure 8** Running times of implementations across different data sets for 2-dimensional convex hull on 36 cores with 2-way hyper-threading.

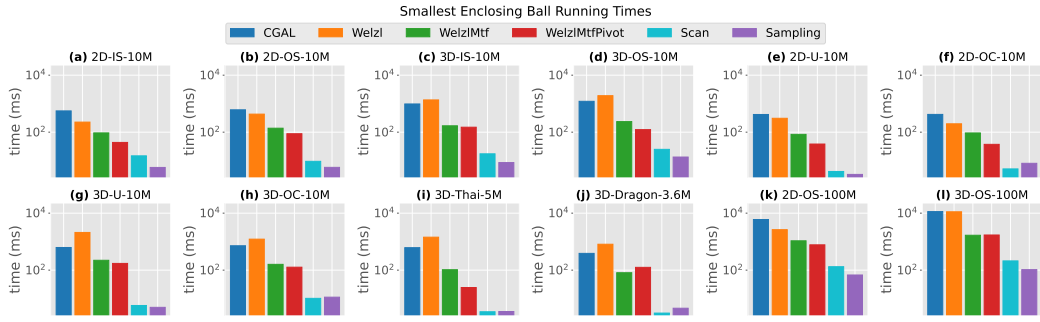


■ **Figure 9** Running times of implementations across different data sets for 3-dimensional convex hull on 36 cores with 2-way hyper-threading.

- **RandInc**: our implementation of the parallel randomized incremental algorithm described in Section 3.
- **QuickHull**: for \mathbb{R}^2 , it is a simple recursive parallel algorithm [19], and we use the implementation in PBBS [46]; for \mathbb{R}^3 , we use our parallel quickhull algorithm described in Section 3.
- **Pseudo**: our implementation of the pseudoHull heuristic proposed by Tang et al. [50] for 3-dimensional convex hull described in Section 3. The final stage of the computation uses our *quickHull* algorithm for \mathbb{R}^3 .
- **DivideConquer**: our divide-and-conquer algorithm described in Section 3.

In Figures 8 and 9, We show a comparison of running times across different methods using 36 cores with two-way hyper-threading. Our implementations achieve significant speedup compared to existing sequential implementations. Our fastest parallel implementations achieve speedups of 190–559x (325x on average) over *CGAL* for 2-dimensional convex hull, and speedups of 10.5–124x (61.4x on average) over *CGAL* for 3-dimensional convex hull. Our fastest parallel implementations have speedups of 147–1673x (605x on average) over 2-dimensional *Qhull*, and speedups of 5.68–43.8x (19.9x on average) over 3-dimensional *Qhull*. When run using a single thread, our parallel implementations achieve speedups of 3.26–12.4x and 1.31–5.05x over *CGAL* for 2 and 3 dimensions, respectively; and 3.39–47.6x and 0.99–2.06x speedups over *Qhull* for 2 and 3 dimensions, respectively.

For \mathbb{R}^2 , *DivideConquer* is always the fastest method due to having high scalability from processing many independent subproblems in parallel. For \mathbb{R}^3 , the fastest two methods are *DivideConquer* and *Pseudo*. We observe that on data sets with a larger output size, *Pseudo* is slower than *DivideConquer* (Figures 9(a), (b), and (g)). This is because the final computation after pruning takes longer given that there are a higher number of remaining points after pruning. For instance, the number of remaining points for *3D-IS-10M* and



■ **Figure 10** Running times of implementations across different data sets for smallest enclosing ball on 36 cores with 2-way hyper-threading.

3D-U-10M after pruning are 83669 and 2316, respectively, and *Pseudo* is relatively slower on the former. We observe that *RandInc* and *QuickHull* take relative longer compared with the fastest methods for data sets with a smaller output size (Figures 9(c)–(e) and (h)). This is caused by higher contention during the reservation of facets, since there are fewer facets on the intermediate hull. For example, for *3D-IS-10M* and *3D-U-10M*, the output sizes are 14163 and 423, respectively. During the computation, *3D-U-10M* exposes fewer facets for reservation, leading to a lower success rate during the reservations.

DivideConquer achieves the best parallel speedup (42.78x and 16.55x on average for \mathbb{R}^2 and \mathbb{R}^3 , respectively). This is because the bulk of the time is spent in computing independent convex hulls across different threads. On the other hand, the incremental algorithms, *RandInc* and *QuickHull*, demonstrate lower scalability because of load imbalance caused by the different amounts of work for each conflict point being processed in parallel.

6.2 Smallest Enclosing Ball

We test the following implementations for smallest enclosing ball (our new implementations are underlined). All implementations work for both \mathbb{R}^2 and \mathbb{R}^3 .

- **CGAL**: sequential C++ implementation of Welzl’s algorithm in CGAL [2].
- **Orthant-scan**: our implementation of Larsson et al.’s orthant-scan algorithm [37].
- **Sampling**: our parallel sampling algorithm described in Section 4.
- **Welzl**: our parallel implementation of Welzl’s algorithm described in Section 4.
- **WelzlMtf**: the same as *Welzl*, but with the move-to-front heuristic [10].
- **WelzlMtfPivot**: the same as *Welzl*, but with both the move-to-front and the pivoting heuristic [32].

For smallest enclosing ball, we show the comparison across implementations using 36 cores with two-way hyper-threading in Figure 10. Our fastest parallel implementations have speedups of 70–178x (109x on average) over *CGAL*. On one thread, our fastest implementations achieve speedup of 2.81–7.05x (4.96x on average) over *CGAL*.

Our sampling-based method is the fastest for eight out of the twelve data sets, whereas *Orthant-scan* without sampling is the fastest for the other four. We observe that the sampling phase on average scans only about 5% of the data set, and results in up to 2.55x (1.47x on average) speedup compared to just running *Orthant-scan*. Comparing across different implementations of Welzl’s algorithms, we see that the move-to-front, and the pivoting heuristic implemented in parallel consistently improve the running times. Specifically, *WelzlMtf* is 2.09–13.9x faster than *Welzl*, and *WelzlMtfPivot* is 3.4–58.6x faster than *Welzl*. We also see that *Sampling* and *Orthant-scan* are 4.63–34.8x and 2.96–40.3x faster than *WelzlMtfPivot*, respectively.

6.3 BDL-tree

We designed a set of experiments to investigate the performance and scalability of BDL-tree and compare it to two baselines that we also implemented. **B1** is a baseline where the kd -tree is rebuilt on each batch insertion and deletion in order to maintain balance. This allows for improved query performance (as the tree is always perfectly balanced) at the cost of slowing down updates. **B2** is another baseline that inserts points directly into the existing tree structure without recalculating the splits. This results in very fast updates at the cost of potentially skewed trees (which slows down query performance). **BDL** is our BDL-tree described in Section 5. We consider splitting the points based on either using the object median (median among the points along a dimension) or the spatial median (splitting the space along a dimension in half).

Construction. Figure 11a shows the scalability of the throughput on the 7D-U-10M data set. As we can see from the results, **BDL** achieves similar or better performance both serially and in parallel than both **B1** and **B2**, and has similar or better scalability than both. With the object median splitting, it achieves up to $34.8\times$ self-relative speedup, with an average self-relative speedup of $28.4\times$. We also note that the single-threaded runtimes are faster with spatial median splitting than with object median splitting. This is because spatial median only involves splitting points at each level compared with finding the median for object median, hence it is less expensive to compute; however, we also note that the scalability for spatial median is lower because there is less work to distribute among parallel threads. The construction of **B2** is significantly slower than that of **B1**, because a separate memory buffer is allocated at each leaf node in **B2** to allow for future insertions. The construction of the BDL-tree is faster than both **B1** and **B2** because splitting the construction across multiple trees while keeping the number of elements the same reduces the total work, and provides ample parallelism when running on multiple threads.

Batch Insertion. In this benchmark, we measure the performance of our batch insertion implementation as compared to the baselines. We measure the time required to insert 10 batches each containing 10% of the points in the data set into an initially empty tree for each of our two baselines as well as our BDL-tree.

From Figure 11b, we see that **B2** achieves the best performance on batched insertions – this is due to the fact that it does not perform any extra work to maintain balance and simply directly inserts points into the existing spatial structure. **BDL** achieves the second-best performance – this is due to the fact that it does not have to rebuild the entire tree on every insert, but amortizes the rebuilding work across the batches. Finally, **B1** has the worst performance, as it must fully rebuild on every insertion. Similar to construction, we note that spatial median splitting performs better in the serial case but has lower scalability. With object median splitting, **BDL** achieves parallel self-relative speedup of up to $35.5\times$, with an average self-relative speedup of $27.2\times$.

Batch Deletion. We measure the time required to delete 10 batches each containing 10% of the points in the data set from an initially full tree for each of our two baselines as well as the BDL-tree. From Figure 11c, we observe that **B2** has vastly superior performance – it does almost no work other than tombstoning the deleted points so it is extremely efficient. Next, we see that **BDL** has the second-best performance, as it amortizes the rebuilding across the batches, rather than having to rebuild across the entire point set for every delete. Finally, **B1** has the worst performance as it rebuilds on every delete. With object median splitting, **BDL** achieves parallel speedup of up to $33.1\times$, with an average speedup of $28.5\times$.

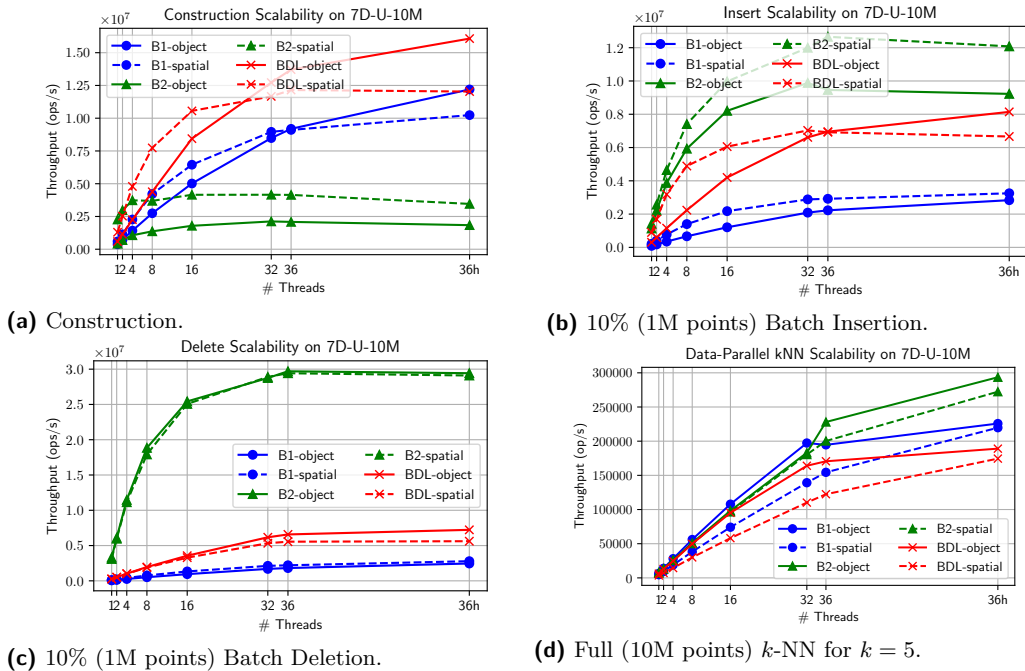


Figure 11 Plot of throughput (operations per second) of batch operations over thread count for both object and spatial median implementations for the 7D-U-10M data set. The prefix of the implementation name refers to the median splitting heuristic. “36h” corresponds to 36 cores with two-way hyper-threading.

Data-Parallel k -NN. We measure the performance and scalability of our k -NN implementation as compared to the baselines. As shown in in Figure 11d, the results show that **B1** and **B2** have similar performance. Furthermore, they are both faster than BDL-tree. This is to be expected, because the k -NN operation is performed directly over the tree after it is constructed over the entire data set in a single batch. Thus, both baselines will consist of fully balanced trees and will be able to perform very efficient k -NN queries. On the other hand, **BDL** consists of a set of multiple trees, which adds overhead to the k -NN operation, as it must be performed separately on each of these individual trees. In the full version of our paper, we show that when the trees are constructed via a set of batch insertions rather than all at once, the performance of **B2** suffers significantly due to the tree being unbalanced.

Comparison with Zd-tree. We compared with the Zd-tree recently proposed by Blelloch and Dobson [21]. The Zd-tree data structure combines the approach of a kd -tree and Morton ordering of the data set, and supports parallel batch-dynamic insertions and deletions, and k -NN. The implementation currently only supports 2 and 3 dimensional data sets, whereas our implementation is not restricted to 2 and 3 dimensions. We tested their implementation on 3D-U-10M. Using all threads, their implementation takes 0.12 seconds to construct, and an average of 0.026 and 0.024 seconds for insertion and deletion of 10% of the data points, and takes 1.65 seconds for k -NN. Our BDL-tree implementation is $3.3\times$, $23.1\times$, and $45.83\times$ slower, for construction, insertion, and deletion, respectively, but achieves roughly the same speed for k -NN search. The reason is that the Morton sort used in their implementation is fast and highly optimized for 2 and 3 dimensions; however, extending this technique to higher dimensions would result in overheads due to more bits needed for the Morton ordering.

7 Conclusion

In this paper, we presented ParGeo, a multicore library for computational geometry containing modules for fundamental tasks including kd -tree based spatial search, spatial graph generation, and algorithms in computational geometry. We also presented new parallel algorithms, implementations, and optimizations for convex hull, smallest enclosing ball, and parallel batch-dynamic kd -tree. We performed a comprehensive experimental study showing that our new implementations achieve significant speedups over prior work and obtain high parallel scalability.

References

- 1 C++ implementation of the 3d quickhull algorithm. <https://github.com/akuukka/quickhull>.
- 2 The computational geometry algorithms library. <https://www.cgal.org/>.
- 3 Header only 3d quickhull in c99. <https://github.com/karimnaaji/3d-quickhull>.
- 4 A header-only C implementation of the quickhull algorithm for building n-dimensional convex hulls and Delaunay meshes. https://github.com/leomccormack/convhull_3d.
- 5 Matlab geometry toolbox for 2d/3d geometric computing. <https://github.com/mattools/matGeom>.
- 6 Qhull. <http://www.qhull.org/>.
- 7 Quickhull3d: A robust 3d convex hull algorithm in Java. <https://www.cs.ubc.ca/~lloyd/java/quickhull3d.html>.
- 8 The Stanford 3d scanning repository. <http://graphics.stanford.edu/data/3Dscanrep/>.
- 9 Pankaj K. Agarwal, Lars Arge, Andrew Danner, and Bryan Holland-Minkley. Cache-oblivious data structures for orthogonal range searching. In *Proceedings of the Symposium on Computational Geometry*, pages 237–245, 2003.
- 10 A. Aggarwal, B. Chazelle, L. Guibas, C. Ó’Dúnlaing, and C. Yap. Parallel computational geometry. *Algorithmica*, 3(1):293–327, March 1988.
- 11 Nancy M. Amato and Franco P. Preparata. The parallel 3d convex hull problem revisited. *International Journal of Computational Geometry & Applications*, 2(02):163–173, 1992.
- 12 Daniel Anderson, Guy E. Blelloch, Laxman Dhulipala, Magdalen Dobson, and Yihan Sun. The problem-based benchmark suite (PBBS), v2. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 445–447, 2022.
- 13 Lars Arge, Gerth Stølting Brodal, and Rolf Fagerberg. Cache-oblivious data structures. In *Handbook of Data Structures and Applications*, pages 545–565. Chapman and Hall/CRC, 2018.
- 14 C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, December 1996.
- 15 Vicente H.F. Batista, David L. Millman, Sylvain Pion, and Johannes Singler. Parallel geometric algorithms for multi-core computers. *Computational Geometry*, 43(8):663–677, 2010.
- 16 Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.
- 17 Jon Louis Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244–251, 1979.
- 18 Jon Louis Bentley and James B Saxe. Decomposable searching problems I. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.
- 19 Guy E Blelloch. *Vector models for data-parallel computing*, volume 2. MIT Press Cambridge, 1990.
- 20 Guy E. Blelloch, Daniel Anderson, and Laxman Dhulipala. ParlayLib - a toolkit for parallel algorithms on shared-memory multicore machines. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures*, pages 507–509, 2020.

- 21 Guy E. Blelloch and Magdalen Dobson. Parallel nearest neighbors in low dimensions with batch updates. In *Proceedings of the Symposium on Algorithm Engineering and Experiments*, pages 195–208, 2022.
- 22 Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, and Julian Shun. Internally deterministic parallel algorithms can be fast. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 181–192, 2012.
- 23 Guy E. Blelloch, Yan Gu, Julian Shun, and Yihan Sun. Parallelism in randomized incremental algorithms. *Journal of the ACM*, 2020.
- 24 Guy E. Blelloch, Yan Gu, Julian Shun, and Yihan Sun. Randomized incremental convex hull is highly parallel. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures*, pages 103–115, 2020.
- 25 Paul B. Callahan and S. Rao Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 291–300, 1993.
- 26 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.
- 27 Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.
- 28 N. Dadoun and D.G. Kirkpatrick. Parallel construction of subdivision hierarchies. *Journal of Computer and System Sciences*, 39(2):153–165, 1989.
- 29 Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.
- 30 Erik D Demaine. Cache-oblivious algorithms and data structures. *Lecture Notes from the EEF Summer School on Massive Data Sets*, 8(4):1–249, 2002.
- 31 Junhao Gan and Yufei Tao. DBSCAN revisited: Mis-claim, un-fixability, and approximation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 519–530, 2015.
- 32 B. Gärtner. Fast and robust smallest enclosing balls. In *European Symposium on Algorithms*, 1999.
- 33 Jonathan S. Greenfield. A proof for a quickhull algorithm, 1990.
- 34 Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. Tetrahedral meshing in the wild. *ACM Trans. Graph.*, 37(4):60:1–60:14, July 2018.
- 35 Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2018. <https://libigl.github.io/>.
- 36 David G. Kirkpatrick and John D. Radke. A framework for computational morphology. In *Computational Geometry*, volume 2 of *Machine Intelligence and Pattern Recognition*, pages 217–248. Springer, 1985.
- 37 Thomas Larsson, Gabriele Capannini, and Linus Källberg. Parallel computation of optimal enclosing balls by iterative orthant scan. *Computers & Graphics*, 56:1–10, 2016.
- 38 D. Lebrun-Grandié, A. Prokopenko, B. Turcksin, and S. R. Slattery. ArborX: A performance portable geometric search library. *ACM Trans. Math. Softw.*, 47(1), December 2020.
- 39 Marco Livesu. cinolib: a generic programming header only C++ library for processing polygonal and polyhedral meshes. *Transactions on Computational Science XXXIV*, 2019. <https://github.com/mlivesu/cinolib/>.
- 40 Kurt Mehlhorn and Stefan Näher. Leda: A platform for combinatorial and geometric computing. *Commun. ACM*, 38(1):96–102, January 1995.
- 41 S. Näher and Daniel Schmitt. A framework for multi-core implementations of divide and conquer algorithms and its application to the convex hull problem. In *Canadian Conference on Computational Geometry (CCCG)*, 2008.
- 42 F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20(2):87–93, February 1977.
- 43 Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

- 44 Octavian Procopiuc, Pankaj K Agarwal, Lars Arge, and Jeffrey Scott Vitter. Bkd-tree: A dynamic scalable kd-tree. In *International Symposium on Spatial and Temporal Databases*, pages 46–65, 2003.
- 45 Julian Shun, Guy E. Blelloch, Jeremy T. Fineman, and Phillip B. Gibbons. Reducing contention through priority updates. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures*, pages 152–163, 2013.
- 46 Julian Shun, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Aapo Kyrola, Harsha Vardhan Simhadri, and Kanat Tangwongsan. Brief announcement: The problem based benchmark suite. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures*, pages 68–70, 2012.
- 47 Daniel Sieger and Mario Botsch. The polygon mesh processing library, 2020. <http://www.pmp-library.org>.
- 48 D Srikanth, Kishore Kothapalli, R Govindarajulu, and P Narayanan. Parallelizing two dimensional convex hull on NVIDIA GPU and Cell BE. In *International Conference on High Performance Computing (HiPC)*, pages 1–5, 2009.
- 49 Ayal Stein, Eran Geva, and Jihad El-Sana. Cudahull: Fast parallel 3d convex hull on the gpu. *Computers & Graphics*, 36(4):265–271, 2012. Applications of Geometry Processing.
- 50 Min Tang, Jie yi Zhao, Ruo feng Tong, and Dinesh Manocha. GPU accelerated convex hull computation. *Shape Modeling International (SMI) Conference*, 36(5):498–506, 2012.
- 51 Stanley Tzeng and John D Owens. Finding convex hulls using quickhull on the GPU. *arXiv preprint arXiv:1201.2936*, 2012.
- 52 Yiqiu Wang, Shangdi Yu, Yan Gu, and Julian Shun. Fast parallel algorithms for Euclidean minimum spanning tree and hierarchical spatial clustering. In *Proceedings of the International Conference on Management of Data*, pages 1982–1995, 2021.
- 53 Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, pages 359–370, 1991.

Combining Predicted and Live Traffic with Time-Dependent A* Potentials

Nils Werner

Karlsruhe Institute of Technology, Germany

Tim Zeitz  

Karlsruhe Institute of Technology, Germany

Abstract

We study efficient and exact shortest path algorithms for routing on road networks with realistic traffic data. For navigation applications, both current (i.e., live) traffic events and predictions of future traffic flows play an important role in routing. While preprocessing-based speedup techniques have been employed successfully to both settings individually, a combined model poses significant challenges. Supporting predicted traffic typically requires expensive preprocessing while live traffic requires fast updates for regular adjustments. We propose an A*-based solution to this problem. By generalizing A* potentials to time dependency, i.e. the estimate of the distance from a vertex to the target also depends on the time of day when the vertex is visited, we achieve significantly faster query times than previously possible. Our evaluation shows that our approach enables interactive query times on continental-sized road networks while allowing live traffic updates within a fraction of a minute. We achieve a speedup of at least two orders of magnitude over Dijkstra's algorithm and up to one order of magnitude over state-of-the-art time-independent A* potentials.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases realistic road networks, shortest paths, live traffic, time-dependent routing

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.89

Related Version *Full Version:* <http://arxiv.org/abs/2207.00381> [21]

Supplementary Material *Software (Source Code):* <https://github.com/kit-algo/tdpot>
archived at `swh:1:dir:f0d810343d53d5ad87db428e89a32cb038dcaeb`

Acknowledgements We want to thank Jonas Sauer for many helpful discussions on algorithmic ideas and proofreading of drafts of this paper. Further, we also want to thank the anonymous reviewers for their helpful comments.

1 Introduction

An important feature of modern routing applications and navigation devices is the integration of traffic information into routing decisions. The more comprehensive the considered traffic information, the better the suggested routes, the more accurate the predicted arrival times and ultimately, the more satisfied the users. For routing, we can distinguish between two aspects of traffic: On the one hand, there are *predictable* traffic flows. For example, certain highways will consistently have traffic jams on weekday afternoons due to commuters driving home. On the other hand, unexpected events such as accidents may also have significant influence on the *current* (i.e., *live*) traffic situation. While it may be sufficient to focus on the current traffic situation to answer short-range routing requests, mid- and long-range queries require taking both types of traffic into account. We therefore aim to provide routing algorithms which incorporate *combined* predicted and live traffic information.



© Nils Werner and Tim Zeitz;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 89; pp. 89:1–89:15
Leibniz International Proceedings in Informatics



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

A common approach for routing in road networks is to model the network as a directed graph where intersections are represented by vertices and road segments by edges. With edge weights representing travel times, routing requests can be answered by solving the classical shortest path problem. When considering predicted traffic, edge weights can be modeled as functions of the time of day, which is commonly referred to as *time-dependent routing*. Dijkstra’s algorithm can be used to solve these problems exactly and, at least from a theoretical perspective, efficiently [8]. However, on the continental-sized road networks used in modern routing applications, it may take seconds to answer mid- or long-range queries, which is too slow for most practical applications. We therefore study algorithms to compute shortest paths significantly faster than Dijkstra’s algorithm while retaining exactness.

One approach to accelerate Dijkstra’s algorithm is goal-directed search, i.e. the A* algorithm [13]. Dijkstra’s algorithm uses a priority queue to explore vertices by ascending distance from the source until it reaches the target. A* changes this slightly and employs a *potential* function which estimates the remaining distance from vertices to the target to change the queue order and explore vertices closer to the target earlier. The performance of A* crucially depends on the tightness of these estimates. The core algorithmic idea of this work is to use A* with time-dependent potential functions, i.e. we obtain tighter estimates and therefore faster queries by taking the time of day when a vertex is visited into account.

1.1 Related Work

Efficient and exact route planning in road networks has received a significant amount of research effort in the past decade. Since a comprehensive discussion is beyond the scope of this paper, we refer to [1] for an overview. An approach that has proven effective is to exploit the fact that usually many queries have to be answered on the same network, which rarely changes. Thus, these queries can be accelerated by computing auxiliary data in an off-line preprocessing phase.

A popular technique which follows this approach is *Contraction Hierarchies* (CH) [9]. During the preprocessing vertices are ranked heuristically by “importance” where more important vertices are part of more shortest paths. Shortcut edges are inserted to skip over unimportant vertices. This allows for a very fast query where only a few vertices are explored. On typical continental-sized networks, queries take well below a millisecond. *Multi-Level Dijkstra* (MLD) [17] is a similar approach that also utilizes shortcut edges but inserts them based on a multi-level partitioning. It achieves slightly slower query times of around a millisecond. MLD is the first algorithm to operate under the *Customizable Route Planning* (CRP) framework [5], i.e. it has a second, faster preprocessing phase called *customization* which allows integrating arbitrary weight functions (or live traffic updates for the current weights) into the auxiliary data without rerunning the entire first preprocessing phase, which is much slower. The MLD customization takes a few seconds, which allows for running it every minute. This three-phase setup has proven to be instrumental to support live traffic scenarios in practical applications [14]. Therefore, CH was generalized to Customizable Contraction Hierarchies (CCH) [7] to support customizability as well.

Both CH and MLD have been extended to time-dependent routing. However, dealing with weight functions instead of scalar weights makes the preprocessing much harder and leads to difficult trade-offs. TCH [2] has fast queries but a very expensive preprocessing phase (up to several hours) and may produce prohibitive amounts of auxiliary data (> 100 GB). TD-CRP [3], an extension of MLD, even follows a three-phase approach and has a relatively fast customization phase. However, this is only possible by giving up exactness. Also, TD-CRP does not support path unpacking. CATCHUp [19] adapts CCH to the time-dependent

setting and has fast and exact queries with significantly reduced memory consumption. While it has a customization phase, running it takes significantly longer than a traditional CCH or CRP customization. On the networks used in this paper, a CATCHUp customization may even take hours, which is too slow for a setting with live traffic updates. Time-dependent Sampling (TDS) [18] is another CH-based approach. While TDS does support both predicted and dynamic traffic information, it cannot guarantee exactness.

ALT [10, 11] is an early A*-based speedup technique for routing in road networks. It combines precomputed distances to a few *landmark* vertices with the triangle inequality to obtain distance estimates to the target vertex. However, query times are significantly slower than with shortcut-based approaches such as CH or MLD. ALT also has been extended to dynamic and time-dependent settings [6]. While this approach allows incremental modifications of the input travel times, it is not as flexible as customization based approaches allowing arbitrary updates.

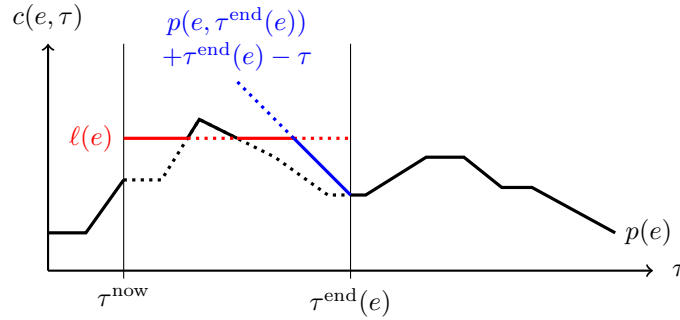
CH-Potentials [20] is another more recent A*-based approach. CH-Potentials use Lazy RPHAST [20], an incremental many-to-one CH query variant, to compute exact distances toward the target. This allows for tighter estimates and faster queries than what is possible with ALT. CH-Potentials can be applied to a variety of routing problem variants. The original publication even mentions a combination of live and predicted traffic. However, the reported query times are above 100 ms. We consider this too slow for practical applications.

1.2 Contribution

In this work, we introduce a time-dependent generalization of A* potentials. We present two Lazy RPHAST extensions that realize a time-dependent potential function and discuss how to apply them to queries in a setting that combines live and predicted traffic. An extensive evaluation confirms the effectiveness of our potentials. Queries incorporating both predicted and current traffic can be answered within few tens of milliseconds. Live traffic updates can be integrated within a fraction of a minute. Our time-dependent potentials are up to an order of magnitude faster than CH-Potentials and about two orders of magnitude faster than Dijkstra's algorithm. To the best of our knowledge, this makes our approach the first to achieve interactive query performance while allowing fast updates in this setting.

2 Preliminaries

We consider simple directed graphs $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges. We use uv as a short notation for an edge from a *tail* vertex u to a *head* vertex v . Weight functions $w : E \rightarrow (\mathbb{Z} \rightarrow \mathbb{N}^0)$ map edges to time-dependent functions, which in turn map a departure time τ at the tail u to a travel time $w(uv)(\tau)$. To simplify notation, we will often write $w(uv, \tau)$. When it is clear from the context that we are writing about constant functions, we omit the time argument and write $w(uv)$. The reversed graph $\overleftarrow{G} = (V, \overleftarrow{E})$ contains a reversed edge vu for every edge $uv \in E$. In this paper, we only need time-independent corresponding reversed weight functions. Therefore, we define $\overleftarrow{w}(vu) = w(uv)$. The travel time of a path $P = (v_1, \dots, v_k)$ is defined recursively $w(P, \tau) = w(v_1v_2, \tau) + w((v_2, \dots, v_k), \tau + w(v_1v_2, \tau))$ with the base case of an empty path having a travel time of zero. A path's travel time can be obtained by successively evaluating travel times of the edges of the path. We denote the travel time of a *shortest* path between vertices s and t for the departure time τ^{dep} as $\mathcal{D}_w(s, t, \tau^{\text{dep}})$. We assume that all travel time functions adhere to the *First-In First-Out* (FIFO) property, i.e. departing later may never lead to an earlier arrival. Formally stated, this means $\tau + w(\tau) \leq \tau + \epsilon + w(\tau + \epsilon)$ for any $\epsilon \geq 0$. With non-FIFO travel time functions, the shortest path problem becomes NP-hard [15, 22].



■ **Figure 1** Combined travel time function $c(e, \tau) = \max(p(e, \tau), \min(\ell(e), p(e, \tau^{\text{end}}(e)) + \tau^{\text{end}}(e) - \tau))$ with both predicted and live traffic information. The predicted traffic $p(e)$ is indicated in black. The live travel time $\ell(e)$ with expected end $\tau^{\text{end}}(e)$ is depicted in red. The switch back to the predicted function is colored in blue. The solid line indicates the combined function $c(e)$ for the current day. For later days, only p will be used. Dotted lines only serve the purpose of visualization.

2.1 Problem Model

We consider an application model with three phases. During the *preprocessing* phase, the graph $G = (V, E)$ and a weight function p of time-dependent traffic predictions are given. Predicted travel time functions are periodic piecewise linear functions represented by a sequence of breakpoints covering one day. A preprocessing algorithm may now precompute auxiliary data, which may take several hours. In the *update* phase, a weight function ℓ of currently observed live travel times are given for the current moment τ^{now} . These live travel times are time-independent and can be represented by a single scalar value. Further, each edge e has a point in time $\tau^{\text{end}}(e)$ when we switch back to the predicted travel time. For edges without live traffic data, we set $\tau^{\text{end}}(e) = \tau^{\text{now}}$. We assume that traffic predictions are conservative estimates and that live traffic will only be slower than the predicted traffic due to accidents and other traffic incidents, i.e. $p(e, \tau^{\text{now}}) < \ell(e)$. Therefore, we define the combined travel time function $c(e, \tau) = \max(p(e, \tau), \min(\ell(e), p(e, \tau^{\text{end}}(e)) + \tau^{\text{end}}(e) - \tau))$. It follows that $p(e, \tau) \leq c(e, \tau)$. The update phase will be repeated frequently and should therefore be as fast as possible. During the final *query* phase, many shortest path queries $(s, t, \tau^{\text{dep}})$ where $\tau^{\text{dep}} \geq \tau^{\text{now}}$ should be answered as quickly as possible by obtaining a path $P = (s, \dots, t)$ that minimizes $c(P, \tau^{\text{dep}})$. Figure 1 depicts an example of such a combined travel time function.

Our model has two important restrictions. First, the dynamic real-time traffic information ℓ is handled separately from the traffic predictions p . Fast updates to the predicted traffic functions p are not the goal of our work. While this might seem less flexible than dynamic traffic predictions p , we believe that our model is actually more practical. This is because the traffic predictions are *periodic* functions. But live traffic incidents are inherently tied to the current moment and are not expected to repeat in 24 hours. Second, our model assumes predicted traffic to be a lower bound of the real-time traffic. If the observed live travel time were faster than the predicted travel time, the live travel time would be ignored. While this a severe restriction from a theoretical perspective, it is only a minor limitation for our practical problem. Live traffic should account for unexpected traffic events which will almost always only make traffic worse. If the live traffic is frequently better than the predicted traffic, the predictions should be adjusted at some point. We discuss these restrictions further and compare our problem to similar models in related work in the full version of this paper [21].

2.2 Fundamental Algorithms

Dijkstra's algorithm [8] computes $\mathcal{D}_w(s, t, \tau^{\text{dep}})$ by exploring vertices in increasing order of distance from s until t is reached. The distances from s to each vertex u are tracked in an array $D[u]$, initially set to ∞ for all vertices. A priority queue of vertices ordered by their distance from s is maintained. The priority queue is initialized with s and $D[s]$ set to τ^{dep} . In each iteration, the next closest vertex u is extracted from the queue and *settled*. Outgoing edges uv are *relaxed*, i.e. the algorithm checks if $D[u] + w(uv, D[u])$ improves $D[v]$. If so, the queue position of v is adjusted accordingly. Once t has been settled, the final distance is known, and the search terminates. We denote visited vertices as the *search space* of a query.

A* [13] is a goal-directed extension of Dijkstra's algorithm. It applies a *potential* function π_t which maps vertices to an estimate of the remaining distance to t . This estimate is added to the queue key. Thus, vertices closer to the target are visited earlier, and the search space becomes smaller. It can be guaranteed that A* has computed the shortest distance once t is settled when the estimates of the potential function are lower bounds of the remaining distances. However, with only lower bound potentials, the theoretical worst case running time of A* is exponential. Therefore, a stronger correctness property is often used. A potential function is called *feasible* if $w(uv) - \pi_t(u) + \pi_t(v) \geq 0$ for any edge $uv \in E$. Feasibility guarantees correctness and polynomial running time. When the potential of the target is zero, i.e. $\pi_t(t) = 0$, it also implies the lower bound property.

Contraction Hierarchies (CH) [9] is a speedup technique to accelerate shortest path searches on time-independent road networks through precomputation. During the preprocessing, a total order $v_1 \prec \dots \prec v_n$ of all vertices $v_i \in V$ by "importance" is determined heuristically, where more important vertices should lie on more shortest paths. Then, an *augmented graph* $G^+ = (V, E^+)$ with additional *shortcut edges* and weights w^+ is constructed. Shortcut edges uv allow to "skip over" paths $(u, \dots, x_i, \dots, v)$ where $x_i \prec u$ and $x_i \prec v$. Therefore, $w^+(uv)$ is assigned the length of the shortest such path. We sometimes split G^+ into an upward graph $G^\uparrow = (V, E^\uparrow)$ which contains only edges uv where $v \succ u$ and a downward graph $G^\downarrow = (V, E^\downarrow)$ defined analogously. The augmented graph has the property that between any two vertices s and t , there exists an *up-down-path* P with $w^+(P) = \mathcal{D}_w(s, t)$ which uses first only edges from E^\uparrow and then only edges from E^\downarrow . Such a path can be found by running the bidirectional variant of Dijkstra's algorithm from s on G^\uparrow and from t on G^\downarrow . Because only a few vertices are reachable in this *CH search space*, queries are very fast, i.e. about a tenth of a millisecond on continental-sized networks.

In this work we build on *Customizable Contraction Hierarchies* (CCH) [7]. For CCH, the construction of the augmented graph is split into two phases. In the first phase, the topology of the augmented graph is constructed without considering any weight functions. It is therefore valid for *all* weight functions. In the second *customization* phase, the weights w^+ of the augmented graph are computed for a given weight function w . The customization can be parallelized efficiently [4] and takes a couple of seconds on typical networks.

Lazy RPHAST [20] is a CH query variant to incrementally compute distances from many sources toward a common target. The first step is to run Dijkstra's algorithm on G^\downarrow from t , similarly to a regular CH query. The second Dijkstra search is replaced with a recursive depth-first search (DFS) which memoizes distances. Algorithm 1 depicts this routine which will be called for all sources. If the distance of a vertex u was previously computed, the routine terminates immediately and returns the memoized value $D[u]$. Otherwise, the distance for all upward neighbors v is obtained recursively. The final distance is the minimum over the path distances $w^+(uv) + D[v]$ via the upward neighbors v and the distance possibly found

■ **Algorithm 1** Computing the distance from a single vertex u to t with Lazy RPHAST.

Data: $D^\downarrow[u]$: tentative distance from u to t computed by Dijkstra's algorithm on G^\leftarrow
Data: $D[u]$: memoized final distance from u to t , initially \perp
Function `ComputeAndMemoizeDist(u):`

```

  if  $D[u] = \perp$  then
     $D[u] \leftarrow D^\downarrow[u]$ ;
    for all edges  $uv \in E^\uparrow$  do
       $D[u] \leftarrow \min(D[u], \text{ComputeAndMemoizeDist}(v) + w^+(uv))$ ;
  return  $D[u]$ ;

```

in the backward search $D^\downarrow[u]$. Using a DFS to compute shortest distances works because G^\uparrow is a directed acyclic graph. Using the distance to t obtained by Lazy RPHAST as an A* potential is called *CH-Potentials*. Just like a regular CH query, Lazy RPHAST can be used on CCH without modifications. In [20] additional optimizations for A* are discussed which we also utilize. The goal is to reduce the impact of the potential evaluation overhead by avoiding unnecessary potential evaluations, for example for chains of degree-two vertices.

3 Time-Dependent A* Potentials

We now propose a time-dependent generalization $\pi_t : V \rightarrow (T \rightarrow \mathbb{Z}^{\geq 0})$ of A* potentials, i.e. estimates are a function of the time. This allows us to obtain tighter estimates and enables faster queries. Analogue to classical potentials, there are properties of time-dependent potentials to consider for the correctness of A*:

- Strong First-In First-Out (FIFO): $\pi_t(v, \tau) < \pi_t(v, \tau + \epsilon) + \epsilon$ for $v \in V$, $\tau > \mathcal{D}_w(s, u, \tau^{\text{dep}})$ and $\epsilon > 0$. This ensures that queue keys increase monotonically with the distance from s . This property has no time-independent equivalent because it holds trivially in this case.
- Feasibility: $w(uv, \tau) + \pi_t(v, \tau + w(uv, \tau)) - \pi_t(u, \tau) \geq 0$ for all edges $uv \in E$ and times $\tau > \mathcal{D}_w(s, u, \tau^{\text{dep}})$. A* can be analyzed as an equivalent run of Dijkstra's algorithm with a modified weight function derived from the input weights and the potentials. With feasibility, these modified weights are non-negative, which implies correctness and polynomial running time. When $\pi_t(t, \tau) = 0$, feasibility also implies the lower bound property. However, feasibility is not strictly necessary to guarantee correctness.
- Lower bound: $\pi_t(v, \tau) \leq \mathcal{D}_w(v, t, \tau)$ for every vertex $v \in V$ and time $\tau = \mathcal{D}_w(s, v, \tau^{\text{dep}})$. This ensures that the search has found the correct distance once the target vertex is settled. This is also sufficient for correctness. However, without feasibility, A* may settle vertices multiple times. In theory, this can lead to an exponential running time.

We discuss these properties in detail and prove the correctness in the full version of this paper [21]. Note that these properties only need to hold for specific times τ , not all possible times of the day. Our practical potentials heavily rely on this and only compute data for the specific times necessary to answer a query correctly.

In the following, we present two practical realizations of time-dependent A* potentials. Both are extensions of Lazy RPHAST. Lazy RPHAST/CH-Potentials is already a very efficient potential and obtains exact distances for scalar lower bound weights, i.e. the tightest possible estimates with a time-independent potential definition. To outperform CH-Potentials, on the one hand, we have to obtain significantly tighter estimates. On the other hand, we

also must avoid the potential evaluation becoming too expensive. Therefore, we avoid costly operations on functions and work with scalar values as much as possible. As a result, even though our potentials are time-dependent, computed estimates during a single query usually will *not* change depending on the visit time of a vertex.

3.1 Multi-Metric Potentials

Let $(s, t, \tau^{\text{dep}})$ be a query and τ^{max} an upper bound on the optimal arrival time at the target. Consider any $\tau' \leq \tau^{\text{dep}}$, $\tau^{\text{max}} \leq \tau''$ and the weight function $l[\tau', \tau''](e) := \min_{\tau' \leq \tau \leq \tau''} p(e, \tau)$. Clearly, $\mathcal{D}_{l[\tau', \tau'']}(v, t)$ provides lower bound estimates of distances to the target vertex during the time relevant for this query. If τ' and τ'' are close to τ^{dep} and τ^{max} and, if the difference between τ' and τ'' is not too big, the estimates will be significantly tighter than global lower bound distances. The *Multi-Metric Potentials* (MMP) approach is based on this observation. Instead of using a single potential based on a global lower bound valid for the entire time, we process multiple lower bound weight functions for different time intervals. At query time, we then select an appropriate weight function. The upper bound τ^{max} is computed with a time-independent CCH query on a scalar upper bound function c_{max}^+ computed during the update phase. Efficiently computing distances with respect to the selected weight function is done with Lazy RPHAST. Therefore, no time-dependent computations need to be performed to evaluate this potential function. MMP only depend on the departure time of the query but not of the potential evaluation time. Still, MMP will be significantly tighter than any time-independent potential can be.

Phase Details. The first step of the preprocessing for this potential is to perform the regular CCH preprocessing, i.e. compute an importance ordering and construct the unweighted augmented graph. Now let I be a set of time intervals. In our implementation, we cover the time between 6:00 and 22:00 with intervals of a length of one, two, four, and eight hours, starting every 30 minutes, and one interval covering the entire day. We do not maintain any additional intervals between 22:00 and 6:00 as most edge weights correspond to their respective free-flow travel time during this period. Thus, the lower bound weights would be equal to the full-day lower bounds. During preprocessing, for each interval $[\tau'_i, \tau''_i] \in I$, we extract lower bound functions $l[\tau'_i, \tau''_i]$ and run the CCH customization algorithm to obtain $l[\tau'_i, \tau''_i]^+$. This can be parallelized trivially. Also, the customization can be parallelized internally. For further engineering details, we refer to [7, 4, 12].

During the update phase, we compute an additional lower bound weight function starting at τ^{now} with duration δ derived from the combined weights c and run the basic customization for it. We use $\delta = 59$ minutes to reasonably cover the live traffic but keep the live interval shorter than any other interval. Further, we extract an upper bound weight function c_{max} which is valid for the entire day for both the predicted and the live traffic, and perform the CCH basic customization to obtain c_{max}^+ .

The query starts with a classical CCH query on the customized upper bound c_{max}^+ to obtain a pessimistic estimate of τ^{max} . We then select the smallest interval $[\tau'_i, \tau''_i]$ such that $[\tau^{\text{dep}}, \tau^{\text{max}}] \subseteq [\tau'_i, \tau''_i]$. Running Lazy RPHAST on G_l^+ with the customized weight function $l[\tau'_i, \tau''_i]^+$ yields the desired potential function. See the full version of this paper [21] for additional optimizations.

Correctness. For any given single query, the estimates obtained by MMP are actually time-independent. They return the exact shortest distances with respect to a lower bound weight function valid for the query. Constant potentials trivially adhere to the strong FIFO property. Also, shortest distances for a lower bound weight function are feasible potentials [20].

3.2 Interval-Minimum Potentials

Interval-Minimum Potentials (IMP) is a time-dependent adaptation of the Lazy RPHAST algorithm. While Lazy RPHAST has a single scalar weight for each edge, the Interval-Minimum Potential uses a time-dependent function. This allows for tighter estimates but introduces new challenges. First, we need an augmented graph with sufficiently accurate time-dependent lower bounds. We utilize the existing CATCHUp customization [19] because it is based on CCH. Second, storing the shortcut travel time functions w^+ may consume a lot of memory. Further, the representation as a list of breakpoints makes the evaluation more expensive than looking up a scalar weight. Therefore, we resort to a different representation and store functions as piecewise constant values in buckets of equal duration. Third, evaluating these functions requires a time argument. While $\pi_t(v, \tau)$ includes the time argument τ for the time at v , Lazy RPHAST also needs a time for every recursive invocation. Therefore, we apply Lazy RPHAST a second time on global upper and lower bound weight functions c_{\max}^+ and p_{\min}^+ to quickly obtain arrival intervals for arbitrary vertices. We then use these intervals to evaluate the edge weights and obtain tight time-dependent lower bounds.

Phase Details. The first preprocessing step is the CCH preprocessing. For the second step, we need to obtain time-dependent travel times for the augmented graph G^+ based on the predicted traffic weights p . For this, we utilize CATCHUp [19], a time-dependent adaptation of CCH. The CATCHUp customization yields for each edge in $uv \in E^+$ approximated *time-dependent* lower bound functions $b^+(uv)$. We transform the time-dependent piecewise linear lower bound functions b^+ into piecewise *constant* lower bound functions $b'^+(uv, \tau) := \min \left\{ b^+(uv, \tau') \mid \beta \lfloor \frac{\tau}{\beta} \rfloor \leq \tau' < \beta (\lfloor \frac{\tau}{\beta} \rfloor + 1) \right\}$ where β is the length of each constant segment. This enables a compact representation. Functions can be represented with a fixed number of values per edge. We use 96 buckets of length $\beta = 15$ minutes. Additionally, we derive a scalar lower bound b_{\min}^+ . Note that b_{\min}^+ is typically tighter than bounds obtained by a time-independent customization on lower bounds of the input functions, i.e. w_{\min}^+ .

In the update phase, we extract a combined traffic upper bound weight function c_{\max} for the entire day and run the CCH customization to obtain c_{\max}^+ .

The query consists of two instantiations of the Lazy RPHAST algorithm. The first one uses the scalar bounds b_{\min}^+ and c_{\max}^+ and computes an interval of possible arrival times at arbitrary vertices when departing from s at τ^{dep} . Since arrival intervals are distances from the source vertex, we have to apply Lazy RPHAST in reverse direction. This means we first run Dijkstra's algorithm from s on G^\uparrow , and then, we apply the recursive distance-memoizing DFS on G^\downarrow for any vertex for which we want to obtain an arrival interval. We denote this instance as AILR for *Arrival Interval Lazy RPHAST*. With these arrival intervals, we can now compute lower bounds to the target with the second Lazy RPHAST instantiation, which uses the time-dependent lower bounds b'^+ . The first step is to run Dijkstra's algorithm from t on G^\downarrow . To relax an edge $uv \in E^\downarrow$, we first need to obtain an arrival interval $[\tau_{\min}, \tau_{\max}]$ at v using AILR. This allows us to determine for vu at the relevant time a tight lower bound $d := \min_{\tau \in [\tau_{\min}, \tau_{\max}]} b'^+(vu, \tau)$. Then, we check if we can improve the lower bound from v to t , i.e. $D^\downarrow[v] \leftarrow \min(D^\downarrow[v], D^\downarrow[u] + d)$. Having established preliminary backward distances for all vertices in the CH search space of t , we can now compute estimates with the recursive distance-memoizing DFS. To obtain a distance estimate for vertex u , we first recursively compute distance estimates $D^\downarrow[v]$ for all upward neighbors v where $uv \in E^\uparrow$. Then, we use AILR to obtain an arrival interval $[\tau_{\min}, \tau_{\max}]$ at u . Finally, we relax the upward edges uv set $D^\downarrow[u] \leftarrow \min(D^\downarrow[u], D^\downarrow[v] + \min_{\tau \in [\tau_{\min}, \tau_{\max}]} b'^+(uv, \tau))$. This yields the final estimate for u .

Choosing a good memory layout for the bucket weights is crucial for the performance. We store all edge weights of each bucket consecutively. Typically, only a few buckets per edge are relevant because the arrival intervals are relatively small. Also, all outgoing edges of each vertex are evaluated consecutively. Thus, having their weights for the same bucket close to each other increases cache hits. See [21] for additional optimizations.

Correctness. Estimates obtained by IMP are lower bounds of the actual time-dependent shortest distances. This directly follows from the correctness of the CATCHUP preprocessing and the Lazy RPHAST algorithm. Also, they do satisfy the strong FIFO property because, for any given single query, the estimates are constant. However, they are not feasible due to the piecewise constant approximation schema. We could not observe any practical negative consequences of this, though.

3.3 Compression

Both of our time-dependent potentials use many weight functions. This can lead to problematic memory consumption. However, since we only need lower bounds, we can merge weight functions. Consider two MMP intervals with weight functions l_1 and l_2 . A combined function $l_{1 \cup 2}(uv) = \min(l_1(uv), l_2(uv))$ is valid for both intervals, albeit less tight. We can merge IMP buckets analogously. Thus, we can reduce memory consumption by trading tightness. Both potentials can handle merged lower bound functions with a layer of indirection: Buckets and intervals are mapped to a weight function ID. The weight of an edge in a merged weight function is the minimum weight of this edge in all included functions.

We now discuss an efficient and well-parallelizable algorithm to iteratively merge weight functions until only k functions remain. In each step, we merge the pair of weight functions with the minimal sum of squared differences of all edge weights. Since comparing all pairs of weight functions is expensive, we track the minimum difference sum Δ_{\min} we have found so far and stop any comparison where the sum exceeds Δ_{\min} . However, even when stopping a comparison, we store the preliminary sum and the edge ID up to which we have summed up the differences. Then, we do not need to start from scratch should we continue to compare this particular pair of weight functions. Finally, we maintain all pairs of weights along with the (possibly preliminary) difference sums in a priority queue ordered by the difference sums. When merging two weight functions, all other associated queue entries are removed from the queue and new entries for comparisons between the new weight function and all other functions are inserted. To determine the next weight function pair to merge, unfinished weight function pairs are popped from the queue and processed in parallel. The minimum difference is tracked in an atomic variable.

4 Evaluation

Environment. Our benchmark machine runs openSUSE Leap 15.3 (kernel 5.3.18), and has 192 GiB of DDR4-2666 RAM and two Intel Xeon Gold 6144 CPUs, each of which has 8 cores clocked at 3.5 GHz and 8×64 KiB of L1, 8×1 MiB of L2, and 24.75 MiB of shared L3 cache. Hyperthreading was disabled and parallel experiments use 16 threads. We implemented our algorithms in Rust¹ and compiled them with `rustc 1.61.0-nightly (c84f39e6c 2022-03-20)` in the release profile with the `target-cpu=native` option.

¹ Our code and experiment scripts are available at <https://github.com/kit-algo/tdpot>.

Datasets. We evaluate our algorithms on two networks for which we have proprietary traffic data available. Sadly, we cannot provide access to these datasets due to non-disclosure agreements. We are not aware of any publicly available real-world traffic feeds or predictions. However, as these datasets are the same ones used in [19, 20], at least some comparability is given. Our first network, PTV Europe, has been provided by PTV² in 2020 and is based on TomTom³ routing data covering Western Europe. It has 28.5M vertices and 61M edges. 76% of the edges have a non-constant travel time. The data includes a traffic incident snapshot from 2020/10/28 07:47 with live speeds and estimated incident durations for 215k vertex pairs. Our second network, OSM Germany, is derived from an early 2020 snapshot of OpenStreetMap and was converted into a routing graph using RoutingKit⁴. It has 16.2M vertices and 35.2M edges. For this instance, we have proprietary traffic data provided by Mapbox⁵. This includes traffic predictions for 38% of the edges in the form of predicted speeds for all five-minute periods over the course of a week. We only use the predictions for one day. Also, we exclude speed values which are faster than the free-flow speed computed by RoutingKit. The data also includes two live traffic snapshots in the form of OSM node ID pairs and live speeds for the edge between the vertices. One is from Friday 2019/08/02 15:41 and contains 320k vertex pairs and the other from Tuesday 2019/07/16 10:21 and contains 185k vertex pairs. The datasets do not contain any estimate for how long the observed live speeds will be valid. We set τ^{end} to one hour after the snapshot time. Note that even though it is smaller and has fewer time-dependent edges, OSM Germany is actually the harder instance. This is because it has more breakpoints per time-dependent edge (124.8 compared to 22.5 on PTV Europe) and the predicted travel times fluctuate more strongly.

Methodology. We evaluate our algorithms by sequentially solving batches of 100k shortest path queries with three different query sets: First, there are *random* queries where source and target are drawn from all vertices uniformly at random. These are mostly long-range queries. Second are *1h* queries where we draw a source vertex uniformly at random, run Dijkstra’s algorithm from it and pick the first node with a distance greater than one hour as the target. Third, we generate queries following the Dijkstra rank methodology [16] to investigate the performance with respect to query distance. For these *rank* queries, we pick a source uniformly at random and run Dijkstra’s algorithm from it. We use every 2^i -th settled vertex as the target for a query of Dijkstra rank 2^i . For queries with only predicted traffic, we pick τ^{dep} uniformly at random. When using live traffic, we set $\tau^{\text{dep}} = \tau^{\text{now}}$. To evaluate the performance of the preprocessing and update phases, we run them 10 and 100 times, respectively. Preprocessing and update phases utilize all cores using 16 threads.

We compare our time-dependent potentials MMP and IMP against time-independent CH-Potentials algorithm realized on CCH. Therefore, we denote this approach as CCH-Potentials. All three potentials use the same CCH vertex order and augmented graph. CCH-Potentials provide heuristic estimates based on a lower bound without any real-time or predicted traffic. Thus, no update phase is necessary to integrate real-time traffic updates. It is the only other speedup technique we are aware of that supports exact queries for our problem model. Dijkstra’s algorithm without any acceleration is our baseline.

² <https://ptvgroup.com>

³ <https://www.tomtom.com>

⁴ <https://github.com/RoutingKit/RoutingKit>

⁵ <https://mapbox.com>

■ **Table 1** Query and preprocessing performance results of different potential functions on different graphs and live traffic scenarios. We report average running times, number of queue pops, relative increases of the result distance over the initial distance estimate and speedups over Dijkstra’s algorithm for 100 k random queries. Additionally, we report preprocessing and update times and the memory consumption of precomputed auxiliary data.

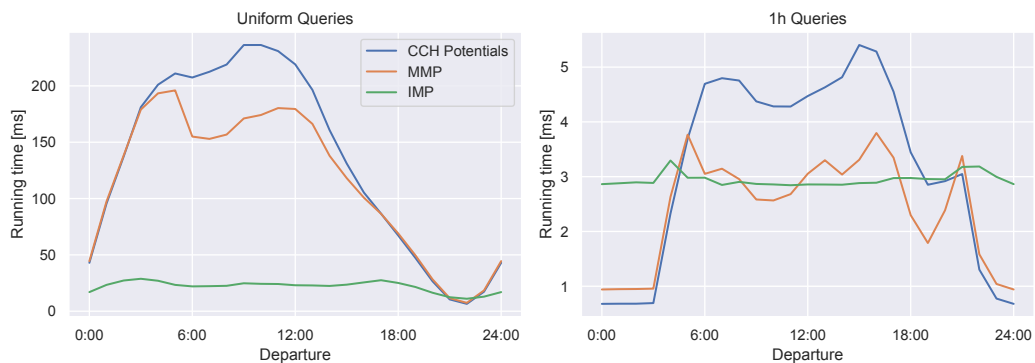
	Graph	Live traffic	Running time [ms]	Queue [$\cdot 10^3$]	Length incr. [%]	Speedup	Prepro. [s]	Update [s]	Space [GB]
CCH Pot.	Ger	–	137.5	92.3	12.2	24.8	–	–	0.8
		10:21	236.5	158.3	18.9	14.7	165.2	–	0.8
		15:41	128.0	89.6	19.1	27.0	–	–	0.8
	Eur	–	102.6	65.2	4.2	58.0	–	–	1.0
		07:47	152.2	102.2	8.4	39.3	249.7	–	1.0
		–	117.7	74.6	9.9	29.0	–	–	33.7
MMP	Ger	10:21	170.0	110.0	13.0	20.4	382.6	15.2	34.0
		15:41	119.0	79.5	15.8	29.0	–	15.3	34.0
		–	95.3	58.6	3.5	62.5	–	–	56.2
	Eur	07:47	131.2	84.5	5.8	45.6	581.5	22.7	57.2
		–	22.2	5.1	1.8	154.1	–	–	30.7
		Ger	10:21	29.1	7.6	2.6	119.2	13 687.0	13.5
15:41	37.7		11.3	4.2	91.5	–	13.6	31.2	
IMP	Eur	–	11.5	1.8	0.4	518.0	–	–	52.1
		07:47	25.4	7.4	1.7	235.5	1 799.9	20.1	53.1

Experiments. In Table 1, we report key performance results for our time-dependent potentials on random queries. We observe that IMP is the fastest approach by a significant margin, up to an order of magnitude faster than time-independent CCH-Potentials and roughly two orders of magnitude faster than Dijkstra’s algorithm. The search space reduction is even greater, but this does not fully translate to running times due to the higher evaluation overhead of IMP. With only predicted traffic, IMP is only two to three times slower than CATCHUp [19]. This shows that using A* to gain algorithmic flexibility comes at a price, but the overhead compared to purely hierarchical techniques is manageable. In contrast, MMP is only slightly faster than CCH-Potentials. This is expected since random queries are mostly long-range for which MMP is not particularly well suited.

Preprocessing times are within a couple of minutes for CCH-Potentials and MMP. IMP preprocessing is significantly more expensive because of the time-dependent CATCHUp preprocessing. This is especially pronounced on OSM Germany where the time-dependent travel time functions fluctuate strongly. Still, running preprocessing algorithms on a daily basis is quite possible. This also underlines that frequently running a CATCHUp customization to include live traffic is not feasible. For both our approaches, real-time traffic updates are possible within a fraction of a minute. MMP is slightly slower because it uses a few more weight functions. Both our approaches are quite expensive in terms of memory consumption, but this can be mitigated through the use of compression (see Figure 4).

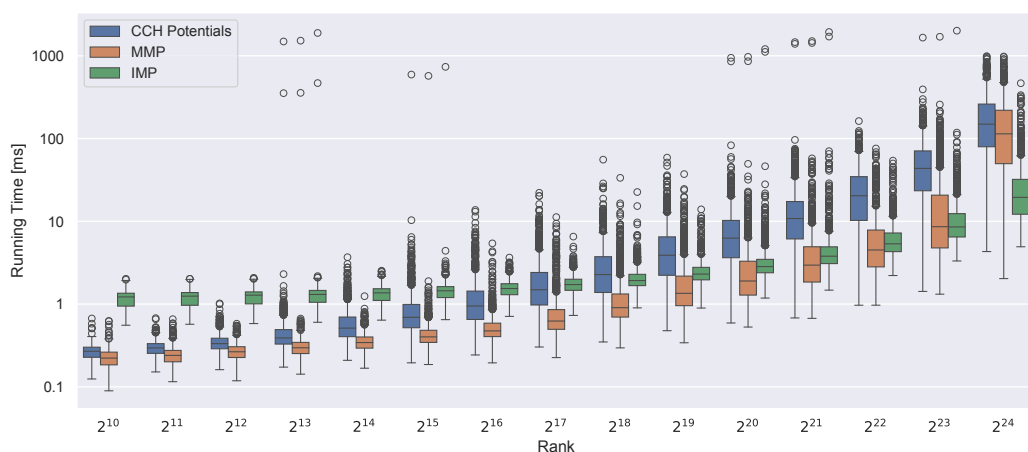
Introducing live traffic decreases the quality of the estimates and thus increases search space sizes and running times. For IMP, this increases running times by roughly a factor of two. Even with heavy rush hour traffic, IMP is still more than 90 times faster than Dijkstra’s algorithm. Surprisingly, for CCH-Potentials and MMP, this scenario seems easier to handle than light midday traffic. This actually is an effect of the *predicted* traffic. It also has a strong influence on the performance of CCH-Potentials and MMP depending on the departure time.

89:12 Combining Predicted and Live Traffic with Time-Dependent A* Potentials

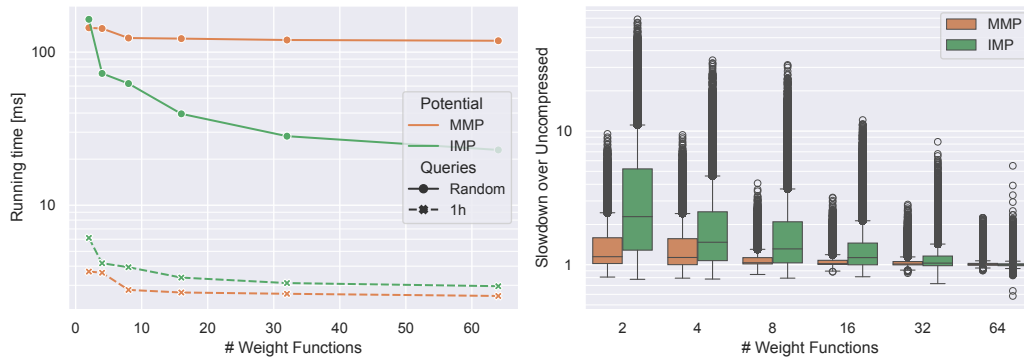


■ **Figure 2** Average running time of 100k uniform and 1h queries on OSM Germany with only predicted traffic. Each query has a departure time drawn uniformly at random. The resulting running times are grouped by the departure time hour.

We investigate this behavior with Figure 2 which depicts query performance by departure time over the course of the day. Clearly, the departure time has a significant influence both for short-range and long-range queries. For long-range queries, the peaks are shifted and smeared because of the travel time (4–5 hours on average on OSM Germany) covered by the query. This is the reason why the heavy afternoon traffic appears to be easier than the light midday traffic for MMP and CCH-Potentials. For IMP, the influence of the departure time is much smaller, which makes it consistently the fastest approach on long-range queries. For short-range queries, the overhead of IMP make it the slowest during the night. Moreover, MMP is roughly as fast as IMP for 1h queries during the daytime. Therefore, MMP may actually be a simple and effective approach for practical applications where short-range queries are more prominent.



■ **Figure 3** Box plot of running times for 1000 queries per Dijkstra-rank on PTV Europe with live traffic and fixed departure at 07:47. The boxes cover the range between the first and third quartile. The band in the box indicates the median; the whiskers cover 1.5 times the interquartile range. All other running times are indicated as outliers.



■ **Figure 4** Left: Mean running times of 100k queries on OSM Germany with only predicted traffic by number of remaining weight functions. Right: Boxplot of the per-query relative slowdown over the running time of the respective query with all weight functions.

Figure 3 depicts the performance by query distance. For short-range queries, IMP is slower than the other approaches because the potential is expensive to evaluate, but it scales much better to long-range queries because of its estimates are tighter. Also, the variance in running times is significantly smaller. Even for rank 2^{24} , most queries can be answered within a few tens of milliseconds. Nevertheless, MMP is actually faster on most ranks. Only at rank 2^{24} , MMP running times become as slow as the CCH-Potentials baseline. A jump in MMP running times can be observed from rank 2^{23} to 2^{24} . This is because the mean query distance jumps from five to six hours on rank 2^{23} to over eight hours on rank 2^{24} , which is longer than the longest covered interval. Thus, on rank 2^{24} , MMP fall back to classical CCH-Potentials on many queries. We also observe a few strong outliers. This happens because of blocked streets in the live traffic data. When the target vertex of a query is only reachable through a blocked road segment, A* will traverse large parts of the networks until the blocked road opens up. This affects all three potentials in the same way and demonstrates an inherent weakness of A*-based approaches: the performance always depends on the quality of the estimates. However, on realistic instances, the time-dependent preprocessing algorithms of purely hierarchical approaches are too expensive for frequent rerunning. This makes our approach the first to enable interactive query times across all distances in a setting with combined live and predicted traffic.

Finally, Figure 4 showcases the effects of reducing the number of weight functions. MMP appears to be very robust against compression. We can reduce the number of weight functions to 16 (a memory usage reduction of about a factor of 6) before the slowdowns become noticeable in the mean running time. However, MMP only achieves relatively small speedups compared to CCH-Potentials, i.e. rarely more than a factor of three. Therefore, its robustness is not particularly surprising. IMP, which achieves stronger speedups, is less robust against compression. Nevertheless, we can reduce the memory consumption by a factor of about three to 32 functions and still achieve very decent query times. With 32 functions, the absolute memory consumption decreases to less than 20 GB, which is at least manageable. Surprisingly, even with only four weight functions, IMP is still faster than MMP on long-range queries. This clearly shows the superiority of IMP for long-range queries. The compression algorithm itself takes less than a minute, depending on the final number of weight functions. Thus, its running time is dominated by the regular preprocessing. See the full paper version [21] for further details on the effectiveness of the parallelization.

5 Conclusion


In this paper, we proposed time-dependent A* potentials for efficient and exact routing in time-dependent road networks with both predicted and live traffic. We presented two realizations of time-dependent potentials with different trade-offs. Both allow fast live traffic updates within a fraction of a minute. IMP achieves query times two orders of magnitude faster than Dijkstra’s algorithm and up to an order of magnitude faster than state-of-the-art time-independent potentials. To the best of our knowledge, this makes our approach the first to achieve interactive query performance while allowing fast updates in this setting. For future work, we would like to apply our time-dependent potentials to other extended scenarios in time-dependent routing, for example to incorporate turn costs.

References

- 1 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016.
- 2 Gernot Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum Time-Dependent Travel Times with Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 18(1.4):1–43, April 2013.
- 3 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Dynamic Time-Dependent Route Planning in Road Networks with User Preferences. In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA’16)*, volume 9685 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2016.
- 4 Valentin Buchhold, Peter Sanders, and Dorothea Wagner. Real-time Traffic Assignment Using Engineered Customizable Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 24(2):2.4:1–2.4:28, 2019. URL: <https://dl.acm.org/citation.cfm?id=3362693>.
- 5 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable Route Planning in Road Networks. *Transportation Science*, 51(2):566–591, 2017. doi:10.1287/trsc.2014.0579.
- 6 Daniel Delling and Giacomo Nannicini. Core Routing on Dynamic Time-Dependent Road Networks. *Informatics Journal on Computing*, 24(2):187–201, 2012.
- 7 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable Contraction Hierarchies. *ACM Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49, April 2016. doi:10.1145/2886843.
- 8 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.
- 9 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, August 2012.
- 10 Andrew V. Goldberg and Chris Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA’05)*, pages 156–165. SIAM, 2005.
- 11 Andrew V. Goldberg and Renato F. Werneck. Computing Point-to-Point Shortest Paths from External Memory. In *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX’05)*, pages 26–40. SIAM, 2005.
- 12 Lars Gottesbüren, Michael Hamann, Tim Niklas Uhl, and Dorothea Wagner. Faster and Better Nested Dissection Orders for Customizable Contraction Hierarchies. *Algorithms*, 12(9):196, 2019. doi:10.3390/a12090196.

- 13 Peter E. Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.
- 14 Bing maps new routing engine. Accessed: 2020-01-25. URL: <https://blogs.bing.com/maps/2012/01/05/bing-maps-new-routing-engine/>.
- 15 Ariel Orda and Raphael Rom. Traveling without waiting in time-dependent networks is NP-hard. Technical report, Dept. Electrical Engineering, Technion-Israel Institute of Technology, 1989.
- 16 Peter Sanders and Dominik Schultes. Highway Hierarchies Hasten Exact Shortest Path Queries. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA '05)*, volume 3669 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2005.
- 17 Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Using Multi-Level Graphs for Timetable Information in Railway Systems. In *Proceedings of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX'02)*, volume 2409 of *Lecture Notes in Computer Science*, pages 43–59. Springer, 2002.
- 18 Ben Strasser. Dynamic Time-Dependent Routing in Road Networks Through Sampling. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASICs)*, pages 3:1–3:17, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASICs.ATMOS.2017.3.
- 19 Ben Strasser, Dorothea Wagner, and Tim Zeitz. Space-efficient, Fast and Exact Routing in Time-Dependent Road Networks. *Algorithms*, 14(3), January 2021. URL: <https://www.mdpi.com/1999-4893/14/3/90>.
- 20 Ben Strasser and Tim Zeitz. A Fast and Tight Heuristic for A* in Road Networks. In David Coudert and Emanuele Natale, editors, *19th International Symposium on Experimental Algorithms (SEA 2021)*, volume 190 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SEA.2021.6.
- 21 Nils Werner and Tim Zeitz. Combining Predicted and Live Traffic with Time-Dependent A* Potentials. Technical report, Institute of Theoretical Informatics, Algorithmics, Karlsruhe Institute of Technology, 2022. arXiv:2207.00381.
- 22 Tim Zeitz. NP-Hardness of Shortest Path Problems in Networks with Non-FIFO Time-Dependent Travel Times. *Information Processing Letters*, May 2022. doi:10.1016/j.ipl.2022.106287.

Approximating Dynamic Time Warping Distance Between Run-Length Encoded Strings

Zoe Xi 

Massachusetts Institute of Technology, Cambridge, MA, USA

William Kuszmaul 

Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

Dynamic Time Warping (DTW) is a widely used similarity measure for comparing strings that encode time series data, with applications to areas including bioinformatics, signature verification, and speech recognition. The standard dynamic-programming algorithm for DTW takes $O(n^2)$ time, and there are conditional lower bounds showing that no algorithm can do substantially better.

In many applications, however, the strings x and y may contain long runs of repeated letters, meaning that they can be compressed using run-length encoding. A natural question is whether the DTW-distance between these compressed strings can be computed efficiently in terms of the lengths k and ℓ of the compressed strings. Recent work has shown how to achieve $O(k\ell^2 + \ell k^2)$ time, leaving open the question of whether a near-quadratic $\tilde{O}(k\ell)$ -time algorithm might exist.

We show that, if a small approximation loss is permitted, then a near-quadratic time algorithm is indeed possible: our algorithm computes a $(1 + \epsilon)$ -approximation for $DTW(x, y)$ in $\tilde{O}(k\ell/\epsilon^3)$ time, where k and ℓ are the number of runs in x and y . Our algorithm allows for DTW to be computed over any metric space (Σ, δ) in which distances are $O(\log n)$ -bit integers. Surprisingly, the algorithm also works even if δ does not induce a metric space on Σ (e.g., δ need not satisfy the triangle inequality).

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis

Keywords and phrases Dynamic time warping distance, approximation algorithms, run-length encodings, computational geometry

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.90

Related Version *Full Version*: <http://arxiv.org/abs/2207.00915>

Funding This research was funded by a Hertz Foundation Fellowship and an NSF GRFP Fellowship. The research was also partially sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator 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.

Acknowledgements The authors would like to thank Charles E. Leiserson for his helpful feedback and suggestions.

1 Introduction

Dynamic Time Warping (DTW) distance is a well-known similarity measure for comparing strings that represent time-series data. DTW distance was first introduced by Vintsyuk in 1968 [40], who applied it to the problem of speech discrimination. In the decades since, DTW has become one of the most widely used similarity heuristics for comparing time series [28] in applications such as bioinformatics, signature verification, and speech recognition [20, 34, 33, 1, 14, 43].



© Zoe Xi and William Kuszmaul;
licensed under Creative Commons License CC-BY 4.0
30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 90; pp. 90:1–90:19
Leibniz International Proceedings in Informatics



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

Consider any two strings x and y , with characters taken from some metric space (Σ, δ) . For example, in many applications, we have that $\Sigma = \mathbb{R}^c$ for some parameter c and that $\delta(a, b) = \|a - b\|_2$ computes ℓ_2 distance. Define a *time warp* of x (and similarly of y) to be any string x' that can be obtained by *warping* the letters in x , where warping a letter means replacing it with ≥ 1 consecutive copies of itself. The DTW-distance $DTW(x, y)$ is defined to be

$$\min_{|x'|=|y'|} \sum_{i=1}^{|x'|} \delta(x'_i, y'_i),$$

where x' and y' range over all time warps of x and y .

The most fundamental question concerning DTW is how to compute it efficiently. Vintsyuk showed that, given strings x and y of length n , it is possible to compute $DTW(x, y)$ in $O(n^2)$ time [40]. His algorithm, which was one of the earliest uses of dynamic programming, continues to be taught in textbooks and algorithms courses today.

For many decades, it was an open question whether any algorithm could achieve a running time of $O(n^{2-\Omega(1)})$. (Interestingly, it is known that one *can* shave small sub-polynomial factors off of the running time [22].) A major breakthrough occurred in 2015, when Abboud, Backurs, and Williams [2] and Bringmann and Künnemann [12] established conditional lower bounds prohibiting any strongly subquadratic-time algorithm for DTW, unless the Strong Exponential Time Hypothesis (SETH) fails.

This lower bound puts us in an interesting situation. On one hand, the classic $O(n^2)$ -time algorithm is often too slow for practical applications. On the other hand, we have good reason to believe that it is nearly optimal. This has led researchers to focus on forms of beyond-worst-case analysis when studying the DTW problem.

An especially appealing question [39, 21] is what happens if x and y both contain long runs of repeated letters. In this case, the strings can be compressed using run-length encoding (RLE). For example, the string “aaaaabbc” has RLE encoding “(a, 5), (b, 2), (c, 1)”. If a string x has k runs, then it is said to have an RLE representation of length k .

It is known that, if x and y each contain k runs, then $DTW(x, y)$ can be computed in $O(k^3)$ time [21].¹ It is still an open question whether it is possible to significantly reduce this cubic running time, and in particular, whether a near-quadratic time algorithm might be possible.

This paper: A Near-Quadratic Approximation Algorithm

We show that, if a small approximation loss is permitted, then a near-quadratic time algorithm is indeed possible. Consider any two run-length encoded strings $x \in \Sigma^n$ and $y \in \Sigma^n$, where x has k runs and y has ℓ runs. Let δ be an arbitrary distance function $\delta : \Sigma \times \Sigma \rightarrow [\text{poly}(n)]$ mapping pairs of characters to $O(\log n)$ -bit nonnegative integers. (Perhaps surprisingly, our algorithms will not require δ to satisfy the triangle inequality, or even to be symmetric.)

Our main result is an algorithm that computes a $(1 + \epsilon)$ -approximation for $DTW(x, y)$ in $\tilde{O}(k\ell/\epsilon^3)$ time². In the special case where Σ is over Hamming space (i.e., $\delta(a, b)$ is either 0 or 1 for all $a, b \in \Sigma$), the running time of our algorithm further improves to $\tilde{O}(k\ell/\epsilon^2)$.

¹ More generally, if x contains k runs and y contains ℓ runs, then the time becomes $O(k\ell^2 + \ell k^2)$.

² Here we are using soft-O notation to mean that $\tilde{O}(k\ell/\epsilon^3)$ is equivalent to $O((k\ell/\epsilon^3) \text{polylog}(n))$.

Our algorithm takes a classical geometric interpretation of DTW in terms of paths through a grid, and shows how to decompose each path in such a way that its components can be efficiently approximated. This allows for us to reduce the problem of approximating DTW-distance between RLE strings to the problem of computing pairwise distance in a small directed acyclic graph.

Other related work

In addition to work on run-length-encoded strings [39, 21], there has been a recent push to study other theoretical facets of the DTW problem. This includes work on approximation algorithms [25, 3, 42], low-distance-regime algorithms [25], communication complexity [11], slightly-subquadratic algorithms [22], reductions to other similarity measures [25, 37, 36], binary DTW [26, 38], etc.

All of these results (along with the results in this paper) can be viewed as part of a larger effort to close the gap between what is known about DTW and what is known about its closely related cousin *edit distance*, which measures the number of insertions, deletions, and substitutions of characters needed to turn one string x into another string y . Like DTW, edit distance can be computed in $O(n^2)$ time using dynamic programming [40, 35] (and can be computed in *slightly* subquadratic time using lookup-table techniques [31]). Also like DTW, edit distance has conditional lower bounds [12, 2, 25] prohibiting strongly subquadratic time algorithms.

When it comes to beyond-worst-case analysis, however, edit distance has yielded much stronger results than DTW: it is known how to compute a constant-approximation for edit distance in strongly subquadratic time [5, 24, 10, 15, 4, 9, 6, 17]; it is known how to compute the edit distance between RLE strings in $\tilde{O}(k\ell)$ time [19, 8, 18, 29, 13, 7, 32, 23, 30]; and if two strings x and y have small edit distance k , it is known how to compute the edit distance in $O(|x| + |y| + k^2)$ time [16, 27].

Whether or not any of these results can be replicated for DTW remains the central open question in modern theoretical work on DTW. There are several reasons to believe that DTW computation should be more challenging than edit distance. Whereas edit distance satisfies the triangle inequality, DTW does not (for example, if we take $\Sigma = \{0, 1\}$, then $DTW(111110, 100000) = 0$, $DTW(100000, 000000) = 1$, and $DTW(111110, 000000) = 5$). This erratic behavior of DTW seems to make it especially difficult to approximate. Additionally, whereas almost all work on edit distance focuses on insertion/deletion/substitution costs of 1, work on DTW must consider arbitrary cost functions δ for comparing characters. Finally, it is known that the problem of computing edit distance actually *reduces* to that of computing DTW [25], indicating that the latter problem is at least as hard (although, interestingly, this reduction *does not* apply in the run-length encoded setting).

Our paper represents the first evidence that an $\tilde{O}(k\ell)$ -time algorithm for DTW may be within reach. Such an algorithm would finally unify edit distance and DTW in the run-length-encoded setting.

2 Technical Overview

This section gives a technical overview of how we approximate DTW -distance between run-length encoded strings. To simplify exposition, we focus here only on the big ideas in the algorithm design and we defer the detailed analysis to later sections.

Throughout the section, we consider two strings x and y of length n whose characters are taken from a set Σ with a symmetric distance function $\delta : \Sigma \times \Sigma \rightarrow \mathbb{N} \cup \{0\}$. Our only assumption on δ is that $\delta(a, b) \in \{0, 1, 2, \dots, \text{poly}(n)\}$ for all $a, b \in \Sigma$. (We do not need the triangle inequality on δ .) Let k and ℓ be the number of runs in x and y , respectively. We will describe a $(1 + O(\epsilon))$ -approximate algorithm that takes $\tilde{O}(k\ell / \text{poly}(\epsilon))$ time.

How to think about DTW

There are several mathematically equivalent ways (see, e.g., [25, 22, 21]) to define the dynamic time warping distance between x and y . In this paper, we work with the geometric interpretation: consider an $n \times n$ grid where cell (i, j) has *cost* $\delta(x_i, y_j)$; consider the *paths* through the grid that travel from $(1, 1)$ to (n, n) via steps of the form $\langle 1, 0 \rangle$ (a horizontal step (h-step) to the right), $\langle 0, 1 \rangle$ (a vertical step (v-step) up), and $\langle 1, 1 \rangle$ (a diagonal step (d-step) to the upper right); the *cost* of such a path is the sum of the costs of the cells that it encounters, and $DTW(x, y)$ is defined to be the smallest cost of any such path. For an example, see Figure 1, which shows an optimal full path for computing $DTW(\text{aaabbbbddd}, \text{aabcbdd}) = 1$, where the δ -function measures the distance between characters in the alphabet.

Note that the i -th column of the grid corresponds to x_i and the j -th row of the grid corresponds to y_j . Thus, each run x_{i_0}, \dots, x_{i_1} in x corresponds to a sequence of adjacent columns i_0, \dots, i_1 in the grid, and each run y_{j_0}, \dots, y_{j_1} in y corresponds to a sequence of adjacent rows j_0, \dots, j_1 in the grid.

If we want to design an algorithm that approximates $DTW(x, y)$ in $\tilde{O}(k\ell / \text{poly}(\epsilon))$ time, then it is natural to think about the grid as follows. We break the grid into *blocks* by drawing a vertical line between every pair of runs in x and a horizontal line between every pair of runs in y ; and label the blocks $\{\mathbf{B}_{i,j}\}_{i \in [k], j \in [\ell]}$, where block $\mathbf{B}_{i,j}$ corresponds horizontally to the i -th run in x and vertically to the j -th run in y . All of the cells within a given block B have the same cost, which we refer to as $\delta(B)$. We may also use $\delta_{i,j}$ for $\delta(\mathbf{B}_{i,j})$. We refer to the first/last row of each block as a lower/upper *horizontal boundary* and to the first/last column of each block as a left/right *vertical boundary*.

Finally, it will be helpful to talk about sequences of blocks that are adjacent horizontally or vertically. An *h-block segment* consists of a sequence of consecutive blocks lined up horizontally. Formally, given $i_1 \leq i_2$ in $[k]$ and j in $[\ell]$, we use $\mathbf{B}_{[i_1, i_2], j}$ for the h-block segment $\mathbf{B}_{i_1, j}, \mathbf{B}_{i_1+1, j}, \dots, \mathbf{B}_{i_2, j}$. Similarly, a *v-block segment* consists of a sequence of consecutive blocks lined up vertically – we use $\mathbf{B}_{i, [j_1, j_2]}$ for the v-block segment $\mathbf{B}_{i, j_1}, \mathbf{B}_{i, j_1+1}, \dots, \mathbf{B}_{i, j_2}$. In the same way that we can talk about the four boundaries of a block, we can talk about the four boundaries of a given h-block or v-block segment.

Intuitively, since there are $O(k\ell)$ blocks, our goal is to design an algorithm that runs in time roughly proportional to the number of blocks.

How to think about the optimal path

Let P be a minimum-cost path through the grid. We can decompose the path into a sequence of disjoint *components* P_1, P_2, P_3, \dots , where each component takes one of two forms:

1. A **horizontal-to-vertical** (h-to-v) component connects a cell on the lower boundary of some v-block segment to another cell on the right boundary of the same v-block segment.
 2. A **vertical-to-horizontal** (v-to-h) component connects a cell on the left boundary of some h-block segment to another cell on the upper boundary of the same h-block segment.
- The components P_1, P_2, P_3, \dots are defined such that the end cell of each P_r connects to the the start cell of each P_{r+1} via a single step (either horizontal, vertical, or diagonal).

We will now describe a series of simplifications that we can make to P while increasing its total cost by at most a $(1 + O(\epsilon))$ -factor. The simplifications are central to the design of our algorithm.

Simplification 1: Rounding each component to start and end on “snap points”

Let us call a grid cell (i, j) an *intersection point* if it lies in the intersection of a horizontal boundary and a vertical boundary. (Each block contains at most four intersection points.) We call a grid cell a *snap point* if either it is an intersection point, or it is of the form $(i + (1 + \epsilon)^t, j)$ on the upper boundary of a block B , or it is of the form $(i + 1 + (1 + \epsilon)^t, j)$ on the lower boundary of a block B , or it is of the form $(i, j + (1 + \epsilon)^t)$ on the right boundary of a block B , or it is of the form $(i, j + 1 + (1 + \epsilon)^t)$ on the left boundary of a block B , where (i, j) is an intersection point of the block B and t is nonnegative integer (since this is a technical overview, we ignore floor and ceiling issues). For each boundary cell p in the grid, define $\text{snap}(p)$ to be the nearest snap point to the right of p , if p is on a horizontal boundary, and to be the nearest snap point above p , if p is on a vertical boundary. If p is on both a horizontal boundary and a vertical boundary, then p is an intersection point, so $\text{snap}(p) = p$.

How much would the cost of P increase if we required each of its components to start and end on snap points? Suppose, in particular, that we replace each component P_r with a component P'_r whose start point p_r has been replaced with $\text{snap}(p_r)$ and whose end point q_r has been replaced with $\text{snap}(q_r)$. It may be that $\text{snap}(q_r)$ does not connect to $\text{snap}(p_{r+1})$, meaning that P'_r and P'_{r+1} do not connect properly. If this happens, however, then one can simply modify the starting-point of P'_{r+1} in order to connect it to P'_r (and it turns out this only makes P'_{r+1} cheaper).

Let P' be the concatenation of P'_1, P'_2, P'_3, \dots . To bound the cost of P' , we can argue that the cost of each P'_r is at most $(1 + \epsilon)$ times that of P_r . To transform P_r into P'_r , the first step is to round the start point p_r of P_r to $\text{snap}(p_r)$ – one can readily see that this only decreases (or leaves unchanged) the cost of P_r . The second step is to round the end point q_r of P_r to $\text{snap}(q_r)$. For simplicity, assume that P_r is an h-to-v component that starts on the lower boundary of some block \mathbf{B}_{i,j_1} and finishes on the right boundary of some block \mathbf{B}_{i,j_2} . Let (u, v) be the lower-right intersection point of \mathbf{B}_{i,j_2} and suppose that P_r finishes in cell $(u, v + s)$. Then P_r incurs cost at least $(s + 1) \cdot \delta_{i,j_2}$ in block \mathbf{B}_{i,j_2} . Moreover, the snap point $\text{snap}(q_r) = \text{snap}(u, v + s)$ is guaranteed to be in the set $\{(u, v + s + t)\}_{t \in \{0, 1, \dots, \epsilon \cdot s\}}$. Thus the cost of traveling from q_r to $\text{snap}(q_r)$ is at most $\epsilon \cdot s \cdot \delta_{i,j_2}$. So the cost of P'_r is at most $(1 + \epsilon)$ times that of P_r .

By analyzing each component in this way, we can argue that $\text{cost}(P') \leq (1 + \epsilon) \text{cost}(P)$. Throughout the rest of the section, we will assume that P has been replaced with P' , meaning that each component starts and ends with a snap point.

Simplification 2: Understanding the structure of each component

Next we observe that each individual component can be assumed to have a relatively simple structure. For simplicity, let us focus on an h-to-v component P_r in a v-block segment $\mathbf{B}_{i,[j_1,j_2]}$. We may assume without loss of generality that all of P_r 's h-steps occur together on the lower boundary of some block; and that all of P_r 's v-steps occur at the end of P_r . In other words, P_r is of the form $D_1 \oplus H \oplus D_2 \oplus U$ where \oplus is for path concatenation, D_1 consists of d-steps, H consists of h-steps (along a lower boundary), D_2 again consists of d-steps, and U consists of v-steps (along a right boundary).³ (See Figure 2 where the path $p_1q_1q_2p_2p_3$ in solid lines is such an example.)

³ Note that the components D_1, H, D_2 , and U are each individually allowed to be length 0.

Combined, these assumptions make it so that P_r is fully determined by four quantities: (1) P_r 's start point p_r , (2) the block $\mathbf{B}_{i,j}$ in which H occurs, (3) the length of H , and (4) the length of U .

Define \overline{P}_r to be the prefix of P_r that terminates as soon as U hits its first snap point. (See Figure 2 where the path $p_1q_1q_2p_2p'_2$ is such a prefix of the path $p_1q_1q_2p_2p_3$.) We will see later that \overline{P}_r is, in some sense, the ‘‘important’’ part of P_r to our algorithm. Observe that \overline{P}_r is fully determined by just three quantities: (1) P_r 's start point p_r , (2) the block $\mathbf{B}_{i,j}$ in which H occurs, and (3) the length of H .

Simplification 3: Reducing the number of options for each component

We will now argue that, if we fix the start point p_r , and we are willing to tolerate a $(1 + O(\epsilon))$ -factor approximation loss, then we only need to consider $\text{poly}(\epsilon^{-1} \log n)$ options for \overline{P}_r .

We begin by considering block $\mathbf{B}_{i,j}$ in which H occurs. Let us define the sequence of blocks B_0, B_1, B_2, \dots so that $B_s = \mathbf{B}_{i,j+s}$ and define the sequence of costs $\delta_0, \delta_1, \delta_2, \dots$ so that $\delta_s = \delta_{i,j+s}$. We say that a block B_s is *extremal* if $(1 + \epsilon)\delta_s \leq \delta_t$ for all $t < s$. If we are willing to tolerate a $(1 + O(\epsilon))$ -factor increase in \overline{P}_r 's cost, then we can assume without loss of generality that H occurs in an extremal block. On the other hand, there are only $O(\log_{1+\epsilon}(n))$ extremal blocks, so this means that we only need to consider $O(\log_{1+\epsilon}(n)) \leq \text{poly}(\epsilon^{-1} \log n)$ options for the starting point of H .

Next we consider the length of the horizontal sub-component H . If we are willing to tolerate a $(1 + O(\epsilon))$ -factor increase in \overline{P}_r 's cost, then we can round $|H|$, the length of H , up to be a power of $(1 + \epsilon)$ (or to be whatever length brings us to the next vertical boundary). Thus we only need to consider $O(\log_{1+\epsilon}(n)) \leq \text{poly}(\epsilon^{-1} \log n)$ options for $|H|$.

Together, the block $\mathbf{B}_{i,j}$ in which H occurs and the length of H fully determine \overline{P}_r . Thus, we have reached the following conclusion: if the start point p_r of the component \overline{P}_r is known, then there are only $\text{poly}(\epsilon^{-1} \log n)$ options that we must consider for what \overline{P}_r could look like. Moreover, although we have considered only h-to-v components here, one can make a similar argument for v-to-h components.

Approximating DTW in $\tilde{O}(kl/\text{poly}(\epsilon))$ time

We will now construct a weighted directed acyclic graph $G = \langle V, E \rangle$ that has two special vertices \mathbf{v}_0 and \mathbf{v}_* and that satisfies the following properties:

- G has a total of $\tilde{O}(kl/\text{poly}(\epsilon))$ vertices/edges, and
- the distance from \mathbf{v}_0 to \mathbf{v}_* in G is a $(1 + O(\epsilon))$ -approximation for $DTW(x, y)$.

This reduces the problem of approximating $DTW(x, y)$ to the problem of computing a distance in a weighted directed acyclic graph. The latter problem, of course, can be solved in linear time with dynamic programming; thus the graph G give us a $\tilde{O}(kl/\text{poly}(\epsilon))$ -time $(1 + O(\epsilon))$ -approximation algorithm for DTW .

We construct G to capture the different ways in which path components P_r can connect together (assuming that the path components take the simplified forms described above). As the vertices $v \in V$ correspond to the snap points p in the grid, we can use a vertex to refer to its corresponding snap point and vice versa. We define \mathbf{v}_0 to be the cell $(1, 1)$ in the grid and \mathbf{v}_* to be the cell (n, n) . We add edges E as follows:

- We connect each snap point p on a horizontal (resp. vertical) boundary to the next snap point q to its right (resp. above it).
- We connect each snap point p on a right (resp. upper) boundary to any snap points q on the adjacent left (resp. lower) boundary that can be reached from p in a single step.
- Each snap point $p \in V$ has $\text{poly}(\epsilon^{-1} \log n)$ out-edges corresponding to the $\text{poly}(\epsilon^{-1} \log n)$ options for what a (truncated) component \overline{P}_r starting at p could look like.⁴

Note that, although we only add edges for *truncated* path components \overline{P}_r (rather than full components P_r), these edges can be combined with edges of the first type in order to obtain the full component. This is why we said earlier that the truncated component is the “important” part of the component.

The paths from \mathbf{v}_0 to \mathbf{v}_* in G correspond to the ways in which we can concatenate path components together to get a full path through the grid; if we assign the appropriate weights to the edges, then the cost of a path through G corresponds to the cost of the same path through the grid. The distance from \mathbf{v}_0 to \mathbf{v}_* is therefore a $(1 + O(\epsilon))$ -approximation for $DTW(x, y)$.

Finally, we must bound the size of G . Each block contains at most four intersection points; so there are $O(k\ell)$ total intersection points. Each intersection point creates at most $O(\log_{1+\epsilon}(n))$ snap points; so there are $O(k\ell\epsilon^{-1} \log n)$ snap points (which are the vertices in V). Each snap point has an out-degree of at most $\text{poly}(\epsilon^{-1} \log n)$. Hence we have:

$$|E| \leq O(k\ell\epsilon^{-1} \log n) \text{poly}(\epsilon^{-1} \log n) = \tilde{O}(k\ell / \text{poly}(\epsilon)).$$

We can therefore compute the distance from \mathbf{v}_0 to \mathbf{v}_* in $\tilde{O}(k\ell / \text{poly}(\epsilon))$ time, as desired.

Paper outline

For the sake of simplicity, there are a number of details that we chose to ignore in this section (such as a time-efficient construction of G and a careful proof that the modifications to P incur only a $(1 + O(\epsilon))$ -factor change in its cost). In the remainder of the paper, we give a formal presentation and analysis of the algorithm outlined above.

3 Preliminaries

We use $[n_1, n_2]$ for the set $\{n_1, n_1 + 1, \dots, n_2\}$ consisting of all the integers between n_1 and n_2 , inclusive, and use $[n]$ as a shorthand for $[1, n]$. We use $T_{m,n}$ for a table consisting of m columns and n rows and $T_{m,n}[i, j]$ for the entry on the i -th column and j -th row, where $(i, j) \in [m] \times [n]$ is assumed. We may use T for $T_{m,n}$ if m and n can be readily inferred from the context. Please note that an entry $T_{m,n}[i, j]$ in a table should be distinguished from the value stored in the entry – when discussing the value, we shall refer to it as the content of the entry $T_{m,n}[i, j]$.

Letters

Let us assume an alphabet Σ , which is possibly infinite. We use δ for a distance function on letters such that $\delta(a, a) = 0$ for any $a \in \Sigma$. We do not require that δ be symmetric or the triangular inequality $\delta(a, c) \leq \delta(a, b) + \delta(b, c)$ hold for δ .

⁴ Note that G is not necessarily simple. If there are multiple ways that a component \overline{P}_r could connect two vertices p_1 and p_2 , then there will be multiple edges from p_1 to p_2 .

(y) -

d	3	3	3	2	2	2	2	0	0	0
d	3	3	3	2	2	2	2	0	0	0
c	2	2	2	1	1	1	1	1	1	1
b	1	1	1	0	0	0	0	2	2	2
a	0	0	0	1	1	1	1	3	3	3
a	0	0	0	1	1	1	1	3	3	3
	a	a	a	b	b	b	b	d	d	d

(x)

■ **Figure 1** An optimal full path of the order (10, 6) whose cost equals 1.

Strings

We use x and y for strings. We write $x = (a_1, \dots, a_m)$ for a string consisting of m letters such that $x[i]$ (often written as x_i), the i -th letter in x , is a_i for each $i \in [m]$. We use a^n for a string of n occurrences of a , which is also referred to as a *run* of a , and \hat{x} for a run-length encoded (RLE) string, which consists of a sequence of runs. We use $|\hat{x}|$ and $\|\hat{x}\|$ for the length and r-length of \hat{x} , which are $m_1 + \dots + m_k$ and k , respectively, in the case $\hat{x} = (a_1^{m_1}, \dots, a_k^{m_k})$.

A run in a string x is maximal if it is not contained in a longer run in x . There is a unique run-length encoding \hat{x} of x that consists of only maximal runs in x , and this encoding \hat{x} is referred to as the RLE representation of x . We also use $\|x\|$ for the number of maximal runs in x (and thus $\|x\| = \|\hat{x}\|$).

We use p for points, which are just integer pairs.

► **Definition 1.** Given a point $p_1 = (i_1, j_1)$, another point $p_2 = (i_2, j_2)$ is a successor of p_1 if (1) $i_2 = i_1 + 1$ and $j_2 = j_1$, or (2) $i_2 = i_1$ and $j_2 = j_1 + 1$, or (3) $i_2 = i_1 + 1$ and $j_2 = j_1 + 1$.

► **Definition 2.** Let $P = \langle p_1, \dots, p_R \rangle$ be a sequence of points such that $p_r \in [m] \times [n]$ holds for each $1 \leq r \leq R$. We call P a path of order (m, n) if p_{r+1} is a successor of p_r for each $1 \leq r < R$ (and this P is sometimes also called a “warping path” [21]). Also, we refer to a path of length 2 as a step that connects a point to one of its successors.

We use $\mathcal{P}(m, n)$ for the set of paths of order (m, n) . A path $P_1 \in \mathcal{P}(m, n)$ is a subpath of another path $P_2 \in \mathcal{P}(m, n)$ if P_1 is contained in P_2 (as a consecutive segment).

► **Definition 3.** Let P_1 and P_2 be two non-empty paths. We use $P_1 \simeq P_2$ to mean that P_1 and P_2 begin at the same point and end at the same point.

► **Definition 4.** Let P_1 and P_2 be two paths such that the first point of P_2 , if it exists, is the successor of the last point of P_1 , if it exists. We write $P_1 + P_2$ to mean the concatenation of P_1 and P_2 (as sequences of points) that forms a path containing both P_1 and P_2 as its subpaths. In the case where both P_1 and P_2 are non-empty, there is a step in $P_1 + P_2$ connecting P_1 and P_2 that consists of the last point in P_1 and the first point in P_2 .

Also, we write $P_1 \oplus P_2$ to mean $P_1 + P'_2$ where the last point of P_1 is assumed to be the first point of P_2 and P'_2 is the tail of P_2 , that is, P'_2 is obtained from removing the first point in P_2 . In other words, $P_1 + P_2$ implies that P_1 and P_2 share no point while $P_1 \oplus P_2$ implies that P_1 and P_2 share one point, which is the last point of P_1 and the first point of P_2 .

► **Definition 5.** Let $x = (a_1, \dots, a_m)$ and $y = (b_1, \dots, b_n)$ be two strings. For each path $P \in \mathcal{P}(m, n)$, there is a value $\text{cost}_{x,y}(P) = \sum_{r=1}^R \delta(a_{i_r}, b_{j_r})$, where P equals $((i_1, j_1), \dots, (i_R, j_R))$. This value is often referred to as the cost of P . We may write $\text{cost}(P)$ for $\text{cost}_{x,y}(P)$ if it is clear from the context what x and y should be.

We call each $P \in \mathcal{P}(m, n)$ a full path if $(i_1, j_1) = (1, 1)$ and $(i_R, j_R) = (m, n)$. The DTW distance between x and y , denoted by $\text{DTW}(x, y)$, is formally defined as the minimum of $\text{cost}_{x,y}(P)$, where P ranges over the set of full paths of order (m, n) . Also, a full path P is referred to as an optimal full path if $\text{cost}_{x,y}(P) = \text{DTW}(x, y)$. Given there are only finitely many paths of order (m, n) , there must exist one full path that is optimal. As an example, the shaded squares in Figure 1 illustrate the following full path of the order $(10, 6)$:

$$\langle (1, 1), (2, 2), (3, 2), (4, 3), (5, 3), (6, 3), (7, 4), (8, 5), (9, 6), (10, 6) \rangle$$

where the number in each square is the assumed distance between the two corresponding letters (computed here as the difference between their positions in the alphabet).

► **Definition 6.** Let $P = \langle (i_1, j_1), \dots, (i_R, j_R) \rangle$.

1. P is a v -path if all the i_r are the same for $1 \leq r \leq R$.
2. P is a h -path if all the j_r are the same for $1 \leq r \leq R$.
3. P is a d -path if $i_{r+1} = i_r + 1$ and $j_{r+1} = j_r + 1$ for $1 \leq r < R$.

Please recall that a step is a path of length 2. If a step is a h -path/ v -path/ d -path, respectively, then it is a h -step/ v -step/ d -step, respectively.

► **Definition 7.** Let $x = (a_1, \dots, a_m)$ and $y = (b_1, \dots, b_n)$ be two strings. We use $T_{\text{DTW}}(x, y)$ for the table $T_{m,n}$ such that the content of $T_{m,n}[i, j]$ is $\delta(a_i, b_j)$ for each $i \in [m]$ and $j \in [n]$.

We may use T_{DTW} for $T_{\text{DTW}}(x, y)$ if x and y can be readily inferred from the context. If a table T_{DTW} can be readily inferred from the context, we often associate a point (i, j) with the entry $T_{\text{DTW}}[i, j]$ and think of a path $P = \langle (i_1, j_1), \dots, (i_R, j_R) \rangle$ as the sequence of entries $T_{\text{DTW}}[i_r, j_r]$ for $1 \leq r \leq R$. As an example, a full path of the order $(10, 6)$ is given in Figure 1, where the path is indicated with the 10 shaded entries.

Suppose that the i th run (j th) in x (y) consists of the letters in x (y) from position i_1 (j_1) to position i_2 (j_2), inclusive. Then there is a corresponding block $\mathbf{B}_{i,j}$ consisting of all the entries $T_{\text{DTW}}[u, v]$ for $i_1 \leq u \leq i_2$ and $j_1 \leq v \leq j_2$. Finally, we introduce notation for discussing specific blocks:

► **Definition 8.** If there exists a block B to the right of $\mathbf{B}_{i,j}$ such that $\delta(B) < \delta(\mathbf{B}_{i,j})$, we use $\beta_h(\mathbf{B}_{i,j})$ for such a B that is the closest to $\mathbf{B}_{i,j}$. In other words, $\beta_h(\mathbf{B}_{i,j})$ is $\mathbf{B}_{i',j}$ for the least i' satisfying $i < i'$ and $\delta(\mathbf{B}_{i',j}) < \delta(\mathbf{B}_{i,j})$. Similarly, if there exists a block B above $\mathbf{B}_{i,j}$ such that $\delta(B) < \delta(\mathbf{B}_{i,j})$, then $\beta_v(\mathbf{B}_{i,j})$ is $\mathbf{B}_{i,j'}$ for the least j' satisfying $j < j'$ and $\delta(\mathbf{B}_{i,j'}) < \delta(\mathbf{B}_{i,j})$.

3.1 Computing DTW Distance with Graphs

It is well known [40] that one can turn the problem of computing $\text{DTW}(x, y)$ for two given strings x and y into a problem of finding the shortest distance between two given vertices in some graph, as follows.

Let $x = (a_1, \dots, a_m)$ and $y = (b_1, \dots, b_n)$. We can construct a directed graph $G_0 = \langle V_0, E_0 \rangle$ such that

1. there is a vertex $\mathbf{v}_{i,j} \in V_0$ for each pair $(i, j) \in [m] \times [n]$, and
2. there is a directed edge $\mathbf{e}(\mathbf{v}_{i_1, j_1}, \mathbf{v}_{i_2, j_2})$ of length $\delta(a_{i_1}, b_{j_1})$ connecting \mathbf{v}_{i_1, j_1} to \mathbf{v}_{i_2, j_2} whenever (i_2, j_2) is a successor of (i_1, j_1) .

90:10 Approximating DTW Distance Between RLE Strings

We use $G_{DTW}(x, y)$ for this graph G_0 and use v and e to range over V_0 and E_0 , respectively. We may also refer to each vertex $\mathbf{v}_{i,j} \in V_0$ simply as point (i, j) if there is no risk of confusion. Clearly, $|V_0|$, the size of V_0 , is mn , and $|E_0|$, the size of E_0 , is bounded by $3mn$ (since each point can have at most 3 successors).

As every warping path is naturally mapped to a path in the graph $G_{DTW}(x, y)$ and vice versa, we can use P to range over both warping paths in $T_{DTW}(x, y)$ and paths in $G_{DTW}(x, y)$ without risking confusion. Given a (non-empty) warping path P in $T_{DTW}(x, y)$, we use $len(P)$ for the length of the corresponding path of P in $G_{DTW}(x, y)$, which equals the cost of P minus the cost associated with the last point in P . Therefore, finding the value of $DTW(x, y)$ is equivalent to finding the shortest distance from $\mathbf{v}_{1,1}$ to $\mathbf{v}_{m,n}$, which can be done by running some version of Dijkstra's shortest distance algorithm. Alternatively, since $G_{DTW}(x, y)$ is acyclic, one can use dynamic programming to find the shortest distance, in which case the running time becomes $O(mn)$. This yields the classic dynamic-programming solution for computing $DTW(x, y)$ [40].

The basic strategy that we use in this paper to design a DTW approximation algorithm can be outlined as follows. Let $G_0 = \langle V_0, E_0 \rangle$ be the graph $G_{DTW}(x, y)$ given above. We try to construct a graph $G = \langle V, E \rangle$ such that $V \subseteq V_0$ holds and the length of each edge e in E that connects a vertex v_1 to another vertex v_2 equals the shortest distance from v_1 to v_2 as is defined in G_0 . Let $dist_0$ and $dist$ be the shortest distance functions on the graphs G_0 and G , respectively. We attempt to prove that

$$dist_0(\mathbf{v}_{1,1}, \mathbf{v}_{m,n}) \leq dist(\mathbf{v}_{1,1}, \mathbf{v}_{m,n}) \leq \alpha \cdot dist_0(\mathbf{v}_{1,1}, \mathbf{v}_{m,n})$$

for some approximation ratio $\alpha > 1$ (e.g., $\alpha = 1 + \epsilon$ for $\epsilon > 0$). By running a shortest-path algorithm on G , we are able to compute $dist(\mathbf{v}_{1,1}, \mathbf{v}_{m,n})$ and thus obtain an α -approximation algorithm for $DTW(x, y)$. As the time complexity of such an algorithm can be bounded by $O(|E|)$ plus the time needed for constructing G , the key to finding a fast algorithm is try to minimize $|E|$, the size of E (while ensuring that the construction of G can be done in $O(|E|)$ time).

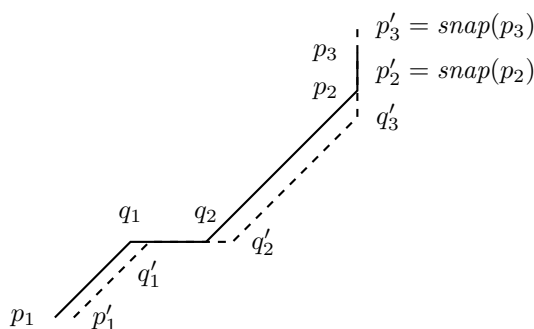
4 A $(1+\epsilon)$ -Approximation Algorithm for DTW

In this section, we present and analyze a $(1 + \epsilon)$ -approximation algorithm for approximating the DTW distance between two run-length encoded strings in near-quadratic time.

Let $x = (a_1, \dots, a_m)$ and $y = (b_1, \dots, b_n)$ be two non-empty strings. Following Section 3.1, our approach will be to construct a graph $G = \langle V, E \rangle$ based on $G_{DTW}(x, y)$, reducing the problem of computing a $(1 + \epsilon)$ -approximation of $DTW(x, y)$ to finding the shortest distance between the vertex $\mathbf{v}_{1,1}$ and the vertex $\mathbf{v}_{m,n}$ in G .

We begin by describing the notions of h-to-v paths and v-to-h paths. The role that these will play in our algorithm is that we will show how to decompose any full path P into a concatenation $P_1 + P_2 + \dots$ of h-to-v and v-to-h paths.

► **Definition 9.** *Let x and y be two non-empty strings. A horizontal-to-vertical (h-to-v) path in $T_{DTW}(x, y)$ is a path that connects a point on the lower boundary of a block B_{i,j_1} to another point on the right boundary of B_{i,j_2} . An h-to-v component in a full path is a maximal h-to-v path that is not contained in any longer h-to-v path in the same full path. A vertical-to-horizontal (v-to-h) path can be defined similarly.*



■ **Figure 2** For illustrating h-to-v path approximation.

Note that an h-to-v path matches characters from a *single run* in x to characters from (possibly multiple) runs in y . We now make the (standard) observation that, when we are comparing a single run to characters to a multi-run string, DTW behaves in a very natural way:

► **Observation 10.** Let $x = (a_1, \dots, a_m)$ and $y = (b_1, \dots, b_n)$ be two non-empty strings. Assume that x is a run of some letter a_0 , that is, $a_0 = a_i$ for $1 \leq i \leq m$.

1. If $m \leq n$, then we have: $DTW(x, y) = \sum_{j=1}^n \delta(a_0, b_j)$. This case corresponds to a path of the form $D \oplus U$, where D consists of only d-steps and U only v-steps.
2. If $m \geq n$, then we have: $DTW(x, y) = \sum_{j=1}^n \delta(a_0, b_j) + (m - n) \cdot \delta(a_0, b_0)$, where b_0 is some b_j closest to a_0 , that is, $\delta(a_0, b_0)$ equals the minimum of $\delta(a_0, b_j)$ for $1 \leq j \leq n$. This case corresponds to a path of the form $D_1 \oplus H \oplus D_2$, where D_1 and D_2 consist of only d-steps and H only h-steps. It should be further noted that, in this case, H can be assumed to travel along the lower boundary of some block, without loss of generality.

We say that a path connecting p and q is optimal if its cost is the least among all the paths connecting p and q . By merging the two cases in Observation 10, we can assume that each optimal h-to-v path is of the form $D_1 \oplus H \oplus D_2 \oplus U$, where any of the four sub-components can vanish. From now on, we can use a 5-tuple $(p_1, q_1, q_2, p_2, p_3)$ (which may also be written as $p_1q_1q_2p_2p_3$) to refer to an h-to-v path, where p_1q_1 is D_1 , q_1q_2 is H , q_2p_2 is D_2 , and p_2p_3 is U . Similarly, each optimal v-to-h path is of the form $D_1 \oplus U \oplus D_2 \oplus H$, and we use a corresponding 5-tuple representation to refer to a v-to-h path as well.

Next we argue that any full path P_0 can be decomposed into h-to-v and v-to-h paths.

► **Lemma 11.** Let x and y be two non-empty strings. Given a full path P_0 in $T_{DTW}(x, y)$, we have $P_0 = P_1 + \dots + P_R$ where P_r is an h-to-v path for each odd $1 \leq r \leq R$ and P_r is a v-to-h path for each even $1 \leq r \leq R$.

Proof. The proof follows directly from the definitions of h-to-v paths and v-to-h paths. For brevity, we defer the full proof to the extended version of the paper [41]. ◀

Given two non-empty strings $x = (a_1, \dots, a_m)$ and $y = (b_1, \dots, b_n)$, we outline as follows a strategy for approximating $DTW(x, y)$. Let P_0 be an optimal full path on $T_{DTW}(x, y)$ such that $cost(P_0) = DTW(x, y)$. By Lemma 11, we have $P_0 = P_1 + \dots + P_R$, where P_1 is an h-to-v path and P_1, \dots, P_R are a sequence of alternating h-to-v paths and v-to-h paths. Let us choose an h-to-v path P_r for some $1 \leq r \leq R$. By Observation 10, we can assume that P_r is of the form of solid lines depicted in Figure 2.⁵

⁵ The meaning of the dashed lines in the figure is to be explained later.

90:12 Approximating DTW Distance Between RLE Strings

In more detail, the path P_r moves diagonally from a point p_1 on the lower boundary of a block B_1 until it meets the lower boundary of another block; it moves horizontally along that lower boundary for some distance; it then moves diagonally to reach a point p_2 on the right boundary of another block B_2 (which is either B_1 or sits above B_1); and finally it moves vertically to reach a point p_3 on the right boundary of another block B_3 (which is either B_2 or sits above B_2). Note that the horizontal moves contained in P_r must be inside a block where those moves cost the least.⁶

Let $G_0 = \langle V_0, E_0 \rangle$ be the graph $G_{DTW}(x, y)$ described in Section 3.1 for computing $DTW(x, y)$. We may use a point (that is, an integer pair) to refer to the corresponding vertex in V_0 . We may also use a path P in $T_{DTW}(x, y)$ to denote its counterpart in G_0 .

Given $\epsilon > 0$, we will construct a graph $G = \langle V, E \rangle$ such that:

- The point $(1, 1)$ is in V and $V \subseteq V_0$ holds.
- Every point in V is on a boundary. Each point in V that is on either a right or upper boundary is connected by an edge to the next snap point on the same boundary (if there is one).
- If there is a step (either an h-step, v-step, or d-step) connecting two boundary points p_1 and p_2 in E_0 , then there is also a step connecting $\text{snap}(p_1)$ and $\text{snap}(p_2)$ in E , where $\text{snap}(p_1)$ (resp. $\text{snap}(p_2)$) is the nearest point in V above or to the right of p_1 (resp. p_2) on the same boundary as p_1 .
- For each h-to-v (resp. v-to-h) path P from p_1 to p_2 (depicted by some solid lines in Figure 2) in G_0 and any point p'_1 in G to the right of p_1 (resp. above p_1) such that p_1 and p'_1 are on the same block boundary, there exists a path P' (depicted by some dashed lines in Figure 2) in G connecting p'_1 and the point $p'_2 = \text{snap}(p_2)$ in G such that $\text{dist}(P') \leq (1 + \epsilon)\text{dist}_0(P)$, where dist and dist_0 are the shortest distance functions on the graphs G and G_0 , respectively.

We remark that our construction of G will repeatedly make use of the following basic fact.

► **Observation 12.** Let $\Delta(t) = \lfloor (1 + \epsilon)^t \rfloor$ for integers $t \geq 0$. For each integer $d \geq 1$, we have a $(1 + \epsilon)$ -approximation of d that is of the form $\Delta(t)$. In other words, $d \leq \Delta(t) \leq (1 + \epsilon) \cdot d$ holds for some t .

We now describe how to construct the graph G . We construct the set V of vertices as follows:

1. Each vertex in G_0 corresponding to a corner point in $T_{DTW}(x, y)$ should be added into V . There are at most $4k\ell$ such vertices, where $k = \|x\|$ and $\ell = \|y\|$.
2. Assume (i, j) is the lower-left corner of block B .
 - If a point $(i + 1 + \Delta(t), j)$ is on the lower boundary of B for some $t \geq 0$, then this point should be added into V . There are at most $\log_{1+\epsilon}(m)$ such points for the block B .
 - If a point $(i, j + 1 + \Delta(t))$ is on the left boundary of B for some $t \geq 0$, then this point should be added into V . There are at most $\log_{1+\epsilon}(n)$ such points for the block B .
3. Assume (i, j) is the upper-left corner of block B . If a point $(i + \Delta(t), j)$ is on the upper boundary of B for some $t \geq 0$, then this point should be added into V . There are at most $\log_{1+\epsilon}(m)$ such points for the block B .
4. Assume (i, j) is the lower-right corner of block B . If a point $(i, j + \Delta(t))$ is on the right boundary of B for some $t \geq 0$, then this point should be added into V . There are at most $\log_{1+\epsilon}(n)$ such points for the block B .

⁶ If there are several blocks in which such horizontal moves can take place, we simply assume that the moves are inside the lowest of these blocks.

The points in V are referred to as **snap points**. Given a snap point p' on the upper or right boundary of some block, if $p'q'$ is a d -step for some point q' , then q' is also a snap point. This can be readily verified by inspecting the construction of V . Also, for each block B , there are at most $O(\log(m+n)/\epsilon)$ points added to V . Therefore, $|V|$, the size of V , is $O(k\ell \cdot \log(m+n)/\epsilon)$ or simply $\tilde{O}(k\ell/\epsilon)$.

► **Definition 13.** *Given a point $p \in V_0$ on a horizontal boundary of a block B , we use $\text{snap}_h(p)$ for the point $p' \in V$ such that p' is p if $p \in V$ or p' is the closest point to the right of p that is on the same boundary of B . The existence of such a point is guaranteed as all of the corner points are included in V . Similarly, $\text{snap}_v(p)$ can be defined for each point p on a vertical boundary of a block.*

We can use $\text{snap}(p)$ for either $\text{snap}_h(p)$ or $\text{snap}_v(p)$ without confusion: If both $\text{snap}_h(p)$ and $\text{snap}_v(p)$ are defined for p , then p must be a corner of some block B , implying $p = \text{snap}_h(p) = \text{snap}_v(p)$ since $p \in V$ holds. We argue as follows that finding $\text{snap}(p)$ for each given p can be done in $\tilde{O}(1)$ time.⁷

► **Definition 14.** *Let $x = (a_1, \dots, a_m)$ be a string and $\hat{x} = (a'_1 \hat{m}_1, \dots, a'_k \hat{m}_k)$ be its RLE representation. Let M_r be $m_1 + \dots + m_r$ for each $0 \leq r < k$. For each $1 \leq i \leq m$, we use \hat{i}_x for the pair (i_0, i_1) such that $i = M_{i_0} + i_1$ for $1 \leq i_1 \leq m_{i_0+1}$.*

We may use \hat{i} for \hat{i}_x if x can be readily inferred from the context.

It is worth taking a moment to verify that we can compute \hat{i}_x efficiently. Assume that an array storing M_r for $0 \leq r < k$ is already built (in $O(k)$ time). Given $i \in [m]$, we can perform binary search on the array to find i_0 in $O(\log(k))$ time such that $M_{i_0} < i \leq M_{i_0+1}$; we can then compute \hat{i}_x as $(i_0, i - M_{i_0})$.

It is also worth verifying that we can compute $\text{snap}(p)$ in $\tilde{O}(1)$ time. Given a point $p = (i, j)$ on a boundary of some block B in $T_{DTW}(x, y)$, we can compute $\hat{i}_x = (i_0, i_1)$ in $O(\log(k))$ time. Similarly, we can compute $\hat{j}_y = (j_0, j_1)$ in $O(\log(\ell))$ time. We can locate the block B as $\mathbf{B}_{i_0+1, j_0+1}$, and then find $\text{snap}(p)$ in $O(1)$ time (assuming $\log_{1+\epsilon}(i_1)$ and $\log_{1+\epsilon}(j_1)$ can be computed in $O(1)$ time). Therefore, given p , we can compute $\text{snap}(p)$ in $\tilde{O}(1)$ time.

Having established that we can compute \hat{i} and $\text{snap}(p)$ efficiently, we are nearly ready to describe the construction of the edges E . Our final task before doing so is to establish a bit more notation for how to talk about blocks.

Please recall that $\beta_h(\mathbf{B}_{i,j})$ (resp. $\beta_v(\mathbf{B}_{i,j})$) refers to the closest block $\mathbf{B}_{i',j}$ (resp. $\mathbf{B}_{i,j'}$) such that $\delta(\mathbf{B}_{i',j}) < \delta(\mathbf{B}_{i,j})$ (resp. $\delta(\mathbf{B}_{i,j'}) < \delta(\mathbf{B}_{i,j})$) holds. If there is no such a block, $\beta_h(\mathbf{B}_{i,j})$ (resp. $\beta_v(\mathbf{B}_{i,j})$) is undefined.

► **Definition 15.** *We refer to $\mathbf{B}_{i_1,j}, \dots, \mathbf{B}_{i_s,j}$ as a β_h -sequence if $\mathbf{B}_{i_{s+1},j} = \beta_h(\mathbf{B}_{i_s,j})$ for $1 \leq s < S$. Let $\beta_h^*(x, y)$ be the length of a longest β_h -sequence. Clearly, we have $\beta_h^*(x, y) \leq \|x\|$. Similarly, we refer to $\mathbf{B}_{i,j_1}, \dots, \mathbf{B}_{i,j_s}$ as a β_v -sequence if $\mathbf{B}_{i,j_{s+1}} = \beta_v(\mathbf{B}_{i,j_s})$ for $1 \leq s < S$. Let $\beta_v^*(x, y)$ be the length of a longest β_v -sequence. Clearly, we have $\beta_v^*(x, y) \leq \|y\|$. Let $\beta^*(x, y) = \max(\beta_h^*(x, y), \beta_v^*(x, y))$.*

Observe that if the underlying distance function δ on letters is from Hamming space, then $\beta^*(x, y) \leq 2$ for any x and y . Slightly more generally, if δ is bounded by a constant, then $\beta^*(x, y)$ is bounded by the same constant plus one. Later in the section, in the proof of Theorem 21, we will also see an important (and much more general) case where $\beta^*(x, y)$ is guaranteed to be $\tilde{O}(1)$.

⁷ We slightly abuse the \tilde{O} notation here as the parameters m and n for the implicit log-terms are not explicitly mentioned.

We are now ready to explain the construction of the set E of edges for connecting vertices in V . The basic idea is to construct E in such a way that, for any constructed path $p'_1 q'_1 q'_2 q'_3 p'_2$ (as is depicted in Figure 2), there should be a path in G going from p'_1 to p'_2 whose cost is at most the cost of the path in G_0 – this allows for the graph G to capture all such paths, and ultimately allows for G to be used in our approximation algorithm. Formally, the construction of E can be performed with the following steps:

1. Note that the corner points in V_0 are all in V . The edges connecting these corner points in E_0 should be added into E .
2. Given a point $p'_1 \in V$ on the upper boundary of a block B , if there is a d-step from p'_1 to p'_2 (on the lower boundary of the block above B), then p'_2 is in V and an edge from p'_1 to p'_2 should be added into E whose length equals $\delta(B)$.
3. Given a point $p'_1 \in V$ on the right boundary of a block B , if there is a d-step from p'_1 to p'_2 (on the left boundary of the block to the right of B), then p'_2 is in V and an edge from p'_1 to p'_2 should be added into E whose length equals $\delta(B)$.
4. If two points $p'_1 = (i_1, j_1)$ and $p'_2 = (i_2, j_2)$ in V are on the same horizontal or vertical boundary of a block B such that p'_2 is the closest point above or to the right of p'_1 , then an edge from p'_1 to p'_2 should be added into E whose length equals $(i_2 - i_1) \cdot \delta(B)$ (horizontal) or $(j_2 - j_1) \cdot \delta(B)$ (vertical).
5. Let p'_1 be a point in V on the lower boundary of a block B_1 . This step adds into E edges between p'_1 and certain chosen snap points q'_4 such that there are h-to-v paths connecting p'_1 and q'_4 .

Let us use B_1^1, \dots, B_1^S for the sequence where $B_1 = B_1^1$ and $B_1^{s+1} = \beta_v(B_1^s)$ for $1 \leq s < S$ and $\beta_v(B_1^S)$ is undefined. Clearly, S is bounded by $\beta_v^*(x, y)$ (according to the definition of $\beta_v^*(x, y)$). Let B'_1 range over B_1^1, \dots, B_1^S .

Let $q'_1 = (i'_1, j'_1)$ be the point on the lower boundary of B'_1 such that the path connecting p'_1 and q'_1 consists of only d-steps. Let $\text{snaps}_h(q'_1)$ be the set consisting of the point q'_1 , the points on the lower boundary of B'_1 of the form $(i'_1 + \Delta(t), j'_1)$ for some $t \geq 0$, and the lower-right corner point of B'_1 . For each q'_2 ranging over the set $\text{snaps}_h(q'_1)$, there exists at most one point q'_3 on the right boundary of some B_2 (which is either B'_1 or sits above B'_1) such that the path connecting q'_2 and q'_3 consists of only d-steps. As this q'_3 may not be in V , we choose q'_4 to be $\text{snap}_v(q'_3)$, which is in V by definition. Note that the path $p'_1 q'_1 q'_2 q'_3 q'_4$ is an h-to-v path in $T_{DTW}(x, y)$. We add into E an edge between p'_1 and q'_4 for each q'_4 . The length of each added edge connecting p'_1 and q'_4 is the shortest distance between p'_1 and q'_4 , which, by Lemma 17, can be computed in $\tilde{O}(1)$ time.

There is one q'_1 for each B'_1 , and there are at most $\log_{1+\epsilon}(m)$ many of q'_2 for each q'_1 , and there is at most one q'_3 for each q'_2 and one q'_4 for each q'_3 . Therefore, for each p'_1 , there are at most $\beta_v^*(x, y) \cdot \log_{1+\epsilon}(m)$ edges added into E .

6. Let p'_1 be a point in V on the left boundary of a block B_1 . This step adds into E edges between p'_1 and certain chosen snap points q'_4 such that there are v-to-h paths connecting p'_1 and q'_4 . We omit the details that are parallel to those in the previous step. There are at most $\beta_h^*(x, y) \cdot \log_{1+\epsilon}(n)$ edges added into E for each p'_1 .

Let us take a moment to discuss how to efficiently compute the lengths of the edges added to E during the construction of $G = \langle V, E \rangle$. That is, how to construct and determine the cost of each dotted path $p'_1 q'_1 q'_2 q'_3 q'_4$ depicted in Figure 2. (Note that $q'_4 = \text{snap}_v(q'_3)$ is not shown in the figure.)

► **Lemma 16.** *Let x and y be two non-empty strings. For $k = \|x\|$ and $\ell = \|y\|$,*

1. *we can compute $\beta_h(B)$ for all the blocks B in $T_{DTW}(x, y)$ in $O(k\ell \cdot \log(k))$ time, and*
2. *we can compute $\beta_v(B)$ for all the blocks B in $T_{DTW}(x, y)$ in $O(k\ell \cdot \log(\ell))$ time.*

Proof. We defer the full proof to the extended version of the paper [41]. ◀

► **Lemma 17.** *For each h-to-v path $(p'_1, q'_1, q'_2, q'_3, q'_4)$, its length can be computed in $\tilde{O}(1)$ time if the five snap points $p'_1, q'_1, q'_2, q'_3,$ and q'_4 are given.*

Proof. We defer the full proof to the extended version of the paper [41]. ◀

For brevity, we omit the obvious lemma parallel to Lemma 17 that is instead on computing the lengths of v-to-h paths in $\tilde{O}(1)$ time.

We are now in a position to state and prove the main theorems of the paper. As noted earlier, the basic idea behind our $(1 + \epsilon)$ -approximation algorithms is to compute a path-distance through the graph $G = (V, E)$, and show that this distance closely approximates $DTW(x, y)$.

We begin by stating a theorem that parameterizes its running time by $\beta^*(x, y)$ – we will then apply this result to obtain fast running times in the cases where the distance function δ outputs either $O(\log n)$ -bit integer values (Theorem 21) or $\{0, 1\}$ -values (Theorem 22).

► **Theorem 18.** *Let $x = (a_1, \dots, a_m)$ and $y = (b_1, \dots, b_n)$ be two non-empty strings, and let \hat{x} and \hat{y} denote the run-length encoded versions of the two strings. There exists a $(1 + \epsilon)$ -approximation algorithm (ApproxDTW) for each $\epsilon > 0$ that takes \hat{x} and \hat{y} as its input and returns a value $\tilde{DTW}(x, y)$ satisfying $DTW(x, y) \leq \tilde{DTW}(x, y) \leq (1 + \epsilon) \cdot DTW(x, y)$. Moreover, the worst-case time complexity of this algorithm is $\tilde{O}(k\ell \cdot \beta^*(x, y)/\epsilon^2)$ for $k = \|x\|$ and $\ell = \|y\|$, where $\beta^*(x, y)$ is defined in Definition 15.*

Proof. The analysis of the approximation ratio follows as in Section 2. ◀

4.1 Time-Bound for Polynomially-Bounded Letter Distances

In this section, we present a variant of the algorithm ApproxDTW for approximating $DTW(x, y)$ under the general condition that the distances between letters are integer values bounded by some polynomial of the lengths of x and y . The time complexity of this variant, which takes \hat{x} and \hat{y} as its input to compute $DTW(x, y)$, is $\tilde{O}(k\ell/\epsilon^3)$ for $k = \|x\|$ and $\ell = \|y\|$.

► **Definition 19.** *Let δ be a distance function on letters such that $\delta(a, b) \geq 1$ if $\delta(a, b) \neq 0$. Given $\epsilon_1 > 0$, we use δ_{ϵ_1} for the distance function such that $\delta_{\epsilon_1}(a, b) = 0$ if $\delta(a, b) = 0$, or $\delta_{\epsilon_1}(a, b) = \text{cpow}(1 + \epsilon_1, \delta(a, b))$ if $\delta(a, b) \geq 1$, where $\text{cpow}(1 + \epsilon_1, \alpha)$ equals $(1 + \epsilon_1)^t$ for the least integer t such that $\alpha \leq (1 + \epsilon_1)^t$ holds.*

Please note that $\delta(a, b) \leq \delta_{\epsilon_1}(a, b) \leq (1 + \epsilon_1) \cdot \delta(a, b)$ holds for any letters a and b .

► **Lemma 20.** *Let $DTW(\delta)$ be the DTW distance function where the underlying distance function for letters is δ . Given $\epsilon_1 > 0$, we have the following inequality for each pair of strings x and y :*

$$DTW(\delta_{\epsilon_1})(x, y) \leq (1 + \epsilon_1) \cdot DTW(\delta)(x, y)$$

Proof. Let P be an optimal full path such that its length based on δ equals $DTW(\delta)(x, y)$. We know that the length of P based on δ_{ϵ_1} is bounded by $(1 + \epsilon_1) \cdot DTW(\delta)(x, y)$ since $\delta_{\epsilon_1}(a, b) \leq (1 + \epsilon_1) \cdot \delta(a, b)$ holds for any letters a and b . As $DTW(\delta_{\epsilon_1})(x, y)$ is bounded by the length of P based on δ_{ϵ_1} , we have the claimed inequality. ◀

90:16 Approximating DTW Distance Between RLE Strings

Let $\epsilon_1 > 0$ and $\epsilon_2 > 0$. By Lemma 20, every $(1 + \epsilon_2)$ -approximation algorithm for DTW based on δ_{ϵ_1} is a $(1 + \epsilon_1) \cdot (1 + \epsilon_2)$ -approximation algorithm for DTW based on δ . For each $\epsilon > 0$, if we choose, for example, $\epsilon_1 = \epsilon/2 - \epsilon^2/2$ and $\epsilon_2 = \epsilon/2$, then we have $(1 + \epsilon_1) \cdot (1 + \epsilon_2) < 1 + \epsilon$, implying that every $(1 + \epsilon_2)$ -approximation algorithm for DTW based on δ_{ϵ_1} is a $(1 + \epsilon)$ -approximation algorithm for DTW based on δ .

► **Theorem 21.** *Let $x = (a_1, \dots, a_m)$ and $y = (b_1, \dots, b_n)$ be two non-empty strings.*

Assume that the underlying distance function δ satisfies (1) $\delta(a, b) \geq 1$ if $\delta(a, b) \neq 0$ and (2) $\delta(a, b)$ is $\text{poly}(m+n)$ for any letters a in x and b in y . There exists a $(1 + \epsilon)$ -approximation algorithm for each $\epsilon > 0$ that takes \hat{x} and \hat{y} as its input and returns a value w satisfying $DTW(x, y) \leq w \leq (1 + \epsilon) \cdot DTW(x, y)$. And the worst-case time complexity of this algorithm is $\tilde{O}(k\ell/\epsilon^3)$ for $k = \|x\|$ and $\ell = \|y\|$.

Proof. Let $\epsilon_1 = \epsilon/2 - \epsilon^2/2$ and $\epsilon_2 = \epsilon/2$. By Theorem 18, ApproxDTW (as is presented in the proof of Theorem 18) takes \hat{x} and \hat{y} as input and returns a $(1 + \epsilon_2)$ -approximation of $DTW(\delta_{\epsilon_1})(x, y)$. And the time complexity of the algorithm is $\tilde{O}(k\ell \cdot \beta^*(x, y)/\epsilon_2^2)$.

Note that there are only $O(\log_{1+\epsilon_1}(m+n))$ -many distinct values of $\delta_{\epsilon_1}(a, b)$ for a and b ranging over letters in x and y , respectively. Hence, for the underlying distance function δ_{ϵ_1} on letters, $\beta_h^*(x, y)$ is $O(\log_{1+\epsilon_1}(m+n))$ and $\beta_v^*(x, y)$ is also $O(\log_{1+\epsilon_1}(m+n))$, which implies that $\beta^*(x, y)$ is $O(\log_{1+\epsilon_1}(m+n))$ or simply $\tilde{O}(1/\epsilon_1)$. Therefore, we can use ApproxDTW to compute a $(1 + \epsilon_2)$ -approximation of $DTW(\delta_{\epsilon_1})(x, y)$ in $\tilde{O}(k\ell/\epsilon_1\epsilon_2^2)$ time. Since any $(1 + \epsilon_2)$ -approximation of $DTW(\delta_{\epsilon_1})(x, y)$ is a $(1 + \epsilon)$ -approximation of $DTW(\delta)(x, y)$, we are done. ◀

4.2 Time-Bound for Constant-Bounded Letter Distances

Assume that there exists a constant N such that $\delta(a, b)$ is an integer less than N for each pair a and b in Σ . For instance, (Σ, δ) satisfies this condition if it is Hamming space (for which N can be set to 2).

► **Theorem 22.** *Assume that $\delta(a, b)$ are $O(1)$ for $a, b \in \Sigma$. Then ApproxDTW, the $(1 + \epsilon)$ -approximation algorithm for DTW given in the proof of Theorem 18, runs in $\tilde{O}(k\ell/\epsilon^2)$ time for $k = \|x\|$ and $\ell = \|y\|$, where \hat{x} and \hat{y} are the input of the algorithm.*

Proof. This theorem follows from Theorem 18 immediately since $\beta^*(x, y)$ is $O(1)$. ◀

5 Conclusion

We have presented in this paper an algorithm for approximating the DTW distance between two RLE strings. Trading accuracy for efficiency, this algorithm is of (near) quadratic-time complexity and thus, as can be expected, asymptotically faster than the exact DTW algorithm of cubic-time complexity [21], which is currently considered the state-of-art of its kind.

It will be interesting to further investigate whether there exist asymptotically faster approximation algorithms for DTW than the one presented in this paper. In particular, it seems both interesting and challenging to answer the open question as to whether there exists a (near) quadratic-time algorithm for computing the (exact) DTW distance between two RLE strings.

References

- 1 John Aach and George M Church. Aligning gene expression time series with time warping algorithms. *Bioinformatics*, 17(6):495–508, 2001.
- 2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for lcs and other sequence similarity measures. In *56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 59–78. IEEE, 2015.
- 3 Pankaj K. Agarwal, Kyle Fox, Jiangwei Pan, and Rex Ying. Approximating dynamic time warping and edit distance for a pair of point sequences. In *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*, pages 6:1–6:16, 2016. doi:10.4230/LIPIcs.SocG.2016.6.
- 4 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 377–386. IEEE, 2010.
- 5 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: It’s a constant factor. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 990–1001. IEEE, 2020.
- 6 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012.
- 7 Alberto Apostolico, Gad M Landau, and Steven Skiena. *Matching for run-length encoded strings*. IEEE, 1997.
- 8 Ora Arbell, Gad M Landau, and Joseph SB Mitchell. Edit distance of run-length encoded strings. *Information Processing Letters*, 83(6):307–314, 2002.
- 9 Ziv Bar-Yossef, T.S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of 45th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 550–559. IEEE, 2004.
- 10 Joshua Brakensiek and Aviad Rubinfeld. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 685–698, 2020.
- 11 Vladimir Braverman, Moses Charikar, William Kuszmaul, David Woodruff, and Lin Yang. The one-way communication complexity of dynamic time warping distance. Manuscript submitted for publication, 2018.
- 12 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 79–97. IEEE, 2015.
- 13 Horst Bunke and János Csirik. Edit distance of run-length coded strings. In *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing: Technological challenges of the 1990’s*, pages 137–143, 1992.
- 14 EG Caiani, A Porta, G Baselli, M Turiel, S Muzzupappa, F Pieruzzi, C Crema, A Malliani, and S Cerutti. Warped-average template technique to track on a cycle-by-cycle basis the cardiac filling phases on left ventricular volume. In *Computers in Cardiology 1998*, pages 73–76. IEEE, 1998.
- 15 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *Journal of the ACM (JACM)*, 67(6):1–22, 2020.
- 16 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th Annual Symposium on Theory of Computing (STOC)*, pages 712–725. ACM, 2016.
- 17 Moses Charikar, Ofir Geri, Michael P Kim, and William Kuszmaul. On estimating edit distance: Alignment, dimension reduction, and embeddings. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

- 18 Kuan-Yu Chen and Kun-Mao Chao. A fully compressed algorithm for computing the edit distance of run-length encoded strings. *Algorithmica*, 65(2):354–370, 2013.
- 19 Raphaël Clifford, Paweł Gawrychowski, Tomasz Kociumaka, Daniel P Martin, and Przemysław Uznanski. Rle edit distance in near optimal time. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- 20 Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. Touch me once and i know it’s you!: implicit authentication based on touch screen patterns. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 987–996. ACM, 2012.
- 21 Vincent Froese, Brijnesh J. Jain, Maciej Rymar, and Mathias Weller. Fast exact dynamic time warping on run-length encoded time series. *CoRR*, abs/1903.03003, 2019. [arXiv:1903.03003](https://arxiv.org/abs/1903.03003).
- 22 Omer Gold and Micha Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10–14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 25:1–25:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. [doi:10.4230/LIPICs.ICALP.2017.25](https://doi.org/10.4230/LIPICs.ICALP.2017.25).
- 23 Guan Shieng Huang, Jia Jie Liu, and Yue Li Wang. Sequence alignment algorithms for run-length-encoded strings. In *International Computing and Combinatorics Conference*, pages 319–330. Springer, 2008.
- 24 Michal Koucký and Michael Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 699–712, 2020.
- 25 William Kuszmaul. Dynamic time warping in strongly subquadratic time: Algorithms for the low-distance regime and approximate evaluation. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9–12, 2019, Patras, Greece*, volume 132 of *LIPICs*, pages 80:1–80:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. [doi: 10.4230/LIPICs.ICALP.2019.80](https://doi.org/10.4230/LIPICs.ICALP.2019.80).
- 26 William Kuszmaul. Binary dynamic time warping in linear time. *arXiv preprint*, 2021. [arXiv:2101.01108](https://arxiv.org/abs/2101.01108).
- 27 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998.
- 28 T. Warren Liao. Clustering of time series data – A survey. *Pattern Recognit.*, 38(11):1857–1874, 2005. [doi:10.1016/j.patcog.2005.01.025](https://doi.org/10.1016/j.patcog.2005.01.025).
- 29 Jia Jie Liu, Guan-Shieng Huang, Yue-Li Wang, and Richard CT Lee. Edit distance for a run-length-encoded string and an uncompressed string. *Information Processing Letters*, 105(1):12–16, 2007.
- 30 Veli Mäkinen, Esko Ukkonen, and Gonzalo Navarro. Approximate matching of run-length compressed strings. *Algorithmica*, 35(4):347–369, 2003.
- 31 William J Masek and Michael S Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, 20(1):18–31, 1980.
- 32 J Mitchell. A geometric shortest path problem, with application to computing a longest common subsequence in run-length encoded strings. *Technical Report, Department of Applied Mathematics, SUNY StonyBrook, NY*, 1997.
- 33 Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. *arXiv preprint*, 2010. [arXiv:1003.4083](https://arxiv.org/abs/1003.4083).
- 34 Mario E Munich and Pietro Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proceedings of 7th International Conference on Computer Vision*, volume 1, pages 108–115, 1999.

- 35 Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.
- 36 Yoshifumi Sakai and Shunsuke Inenaga. A faster reduction of the dynamic time warping distance to the longest increasing subsequence length. *arXiv preprint*, 2020. [arXiv:2005.09169](#).
- 37 Yoshifumi Sakai and Shunsuke Inenaga. A reduction of the dynamic time warping distance to the longest increasing subsequence length. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 38 Nathan Schaar, Vincent Froese, and Rolf Niedermeier. Faster binary mean computation under dynamic time warping. *arXiv preprint*, 2020. [arXiv:2002.01178](#).
- 39 Anooshiravan Sharabiani, Houshang Darabi, Samuel Harford, Elnaz Douzali, Fazle Karim, Hereford Johnson, and Shun Chen. Asymptotic dynamic time warping calculation with utilizing value repetition. *Knowl. Inf. Syst.*, 57(2):359–388, 2018. doi:10.1007/s10115-018-1163-4.
- 40 Taras K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.
- 41 Zoe Xi and William Kuszmaul. Approximating dynamic time warping distance between run-length encoded strings. *CoRR*, abs/2207.00915, 2022. [arXiv:2207.00915](#).
- 42 Rex Ying, Jiangwei Pan, Kyle Fox, and Pankaj K Agarwal. A simple efficient approximation algorithm for dynamic time warping. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 21. ACM, 2016.
- 43 Yunyue Zhu and Dennis Shasha. Warping indexes with envelope transforms for query by humming. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 181–192. ACM, 2003.

Correlated Stochastic Knapsack with a Submodular Objective

Sheng Yang ✉ 

Northwestern University, Evanston, IL, USA

Samir Khuller ✉

Northwestern University, Evanston, IL, USA

Sunav Choudhary ✉

Adobe Research, Bangalore, India

Subrata Mitra ✉

Adobe Research, Bangalore, India

Kanak Mahadik ✉

Adobe Research, San Jose, CA, USA

Abstract

We study the correlated stochastic knapsack problem of a submodular target function, with optional additional constraints. We utilize the multilinear extension of submodular function, and bundle it with an adaptation of the relaxed linear constraints from Ma [Mathematics of Operations Research, Volume 43(3), 2018] on correlated stochastic knapsack problem. The relaxation is then solved by the stochastic continuous greedy algorithm, and rounded by a novel method to fit the contention resolution scheme (Feldman et al. [FOCS 2011]). We obtain a pseudo-polynomial time $(1 - 1/\sqrt{e})/2 \simeq 0.1967$ approximation algorithm with or without those additional constraints, eliminating the need of a key assumption and improving on the $(1 - 1/\sqrt[4]{e})/2 \simeq 0.1106$ approximation by Fukunaga et al. [AAAI 2019].

2012 ACM Subject Classification Theory of computation → Rounding techniques; Theory of computation → Stochastic approximation; Theory of computation → Stochastic control and optimization; Theory of computation → Submodular optimization and polymatroids

Keywords and phrases Stochastic Knapsack, Submodular Optimization, Stochastic Optimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.91

Related Version *Full Version*: <https://arxiv.org/abs/2207.01551> [30]

Funding Samir Khuller and Sheng Yang were funded by an Amazon Research Award, and Adobe Research.

Sheng Yang: Part of the work was done while at University of Maryland, College Park.

Acknowledgements We would like to thank the reviewers for comments on a previous version of this draft, who found a flaw in the algorithm and its analysis. The old algorithm imposes the partition matroid *while* doing the first rounding step, leading to dependency between items. This seemingly convenient step actually breaks the correctness of contention resolution scheme, which is built on FKG inequality and intrinsically needs an independent rounding step. We fixed the issue by replacing it with a true independent rounding step, and fix the solution to fit the partition matroid later on. While this breaks the symmetry between items, the gap of 2 turns out to be large enough to fix everything. Check the use of union bound in Case 1 for the proof of Lemma 10 for details.

1 Introduction

The knapsack problem is one of the most celebrated frameworks to model profit maximization with limited resources. Though well understood in its basic form, many new variants were proposed to model and target more complicated problems. One line of variants take



© Sheng Yang, Samir Khuller, Sunav Choudhary, Subrata Mitra, and Kanak Mahadik; licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 91; pp. 91:1–91:14

Leibniz International Proceedings in Informatics



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

randomness into consideration. Such randomness may appear on item sizes only, on item profits only, or on both in a correlated fashion. A significant body of work [4, 10, 22, 25, 26] connects knapsack problem with the field of stochastic optimization, greatly broadening the spectrum of knapsack problems while introducing various challenges for theoretical analysis. Another line of variants model diminishing returns in the profit, leading to the field of submodular optimization [27, 14, 6, 7, 12, 13], which enjoys tremendous popularity both in theory and in practice. The two lines of work are connected together into stochastic submodular optimization, another fruitful field [3, 20, 9, 15, 17, 18, 21, 31]. In this work, we follow this line, and consider a correlated stochastic knapsack problem with a submodular target function. We arrived at this problem when modeling the spot scheduling problem in Yang et al. [29] (see details in the full version [30]). A slight variant of the final problem was first proposed in Fukunaga et al. [16], trying to model “performance-dependent costs of items” in stochastic submodular optimization. This problem turns out to be a very powerful framework that applies to several other real world applications, like recommendation systems [32, 1], and batch-mode active learning [23].

1.1 Formal Problem Statement

There are n items, each takes a random size $\text{size}_i \in \mathbb{N}$ with probability $p_i(\text{size}_i)$, and gets a reward that *corresponds* to its size. In other words, for each item i , there is a reward function $R_i : \mathbb{N} \rightarrow [M]$, such that $r_i = R_i(\text{size}_i)$. (For simplicity, we define $[n]$ to be the set $\{0, 1, 2, \dots, n\}$, and M a positive integer that upper bounds the maximum reward.) We assume each R_i to be non-decreasing, i.e., the larger an item, the more reward it deserves. We are given a budget $B \in \mathbb{N}$ for the total size of items, and wish to extract as much profit as possible. The total profit is a lattice-submodular function¹ $f : [M]^n \rightarrow \mathbb{R}^+$ on the rewards of included items, and we wish to maximize its expectation².

Items are put in the knapsack one by one. As soon as an item is put in the knapsack, its reward and size are revealed. We halt when the knapsack overflows (not collecting the last item’s reward), and proceed to add another item otherwise. We consider adaptive policies, i.e., we can choose an item to include, observe its realized size, and make further decisions based on the realized size. At first, only the reward function and the size distribution of items are known. When the policy includes an item i , its size_i is realized, and so is its reward $r_i = R_i(\text{size}_i)$. In this work, we only consider adaptive policies without cancellation, i.e., the policy can make its decision based on all the realizations it has seen so far, and the inclusion of an item is irrevocable.

For a vector $q \in [M]^n$, let $\Pr_\gamma[q]$ denote the probability that we get outcome q when running policy γ . Note this probability is with respect to the randomness both in the state of items and in the policy γ . Let $f_{\text{avg}}(\gamma)$ denote $\sum_{q \in [M]^n} \Pr_\gamma[q] f(q)$, i.e., the average objective value obtained by γ . Our aim is to find a policy γ that maximizes $f_{\text{avg}}(\gamma)$. We say γ is an α -approximation policy if $f_{\text{avg}}(\gamma) \geq \alpha f_{\text{avg}}(\gamma^*)$ for any policy γ^* .

In addition to all the above, we further require that the chosen set of items S be an independent set of a partition matroid³ $\mathcal{I} = \{\mathcal{I}_k\}_{k \in [K]}$. This is without loss of generality as we can put each item in a separate partition, and every subset of items is valid. The

¹ See definition of partition matroid, submodular and lattice-submodular in Section 3.

² Let $S \subseteq [n]$, we sample a vector $q \in [M]^n$ as follows. Each component $q(i)$ is sampled independently. For $i \in S$, $\Pr[r_i = R_i(s)] = p_i(s)$; for $i \notin S$, $r_i = 0$ with probability 1. Denote this distribution as q_S . Then the objective is to select a (random) set $S \subseteq \mathcal{I}$ of items that maximizes $\mathbb{E}_{\theta \sim q_S}[f(\theta)]$ subject to $\sum_{i \in S} \text{size}_i \leq B$.

³ See definition of partition matroid, submodular and lattice-submodular in Section 3.

additional constraint allows us to impose conflicts between items, which is needed for the modeling in Yang et al. [29]. More importantly, it is also crucial if we are to allow the attempt to include an item that could possibly overflow the knapsack, a case unsolved and left as open problem in Fukunaga et al. [16] (see details in Section 1.3). This partition matroid is also used to ensure the correctness of our approach based on a time-indexed LP.

1.2 Our Contributions

We present a pseudo-polynomial time algorithm for the correlated stochastic knapsack problem with a submodular target function. It computes an adaptive policy for this problem which is guaranteed to achieve $(1 - 1/\sqrt{\epsilon})/2 \simeq 0.1967$ of the optimal solution on expectation. It improves on the $(1 - 1/\sqrt[4]{\epsilon})/2 \simeq 0.1106$ approximation algorithm from Fukunaga et al. [16]. Furthermore, we eliminate one key assumption in Fukunaga et al. [16] which does not allow the inclusion of any item which could possibly overflow the budget.

1.3 Eliminating An Assumption in Previous Work

In Fukunaga et al. [16], the authors considered a slightly different problem. They made two assumptions, and we managed to eliminate one of them. The first assumption states that larger size means larger reward for every particular job. This assumption is reasonable for general problems and remains crucial in our analysis. The second assumption states that we will never select an item which could overflow the budget, given the realization of selected items⁴. However, for many cases, selecting such an item is a desirable choice since additional value is obtained with high probability. If we are unlucky and the size goes beyond the remaining budget, we either receive a partial value, or do not get any value at all.

1.4 Our Techniques

If the target function is linearly additive, this problem becomes the correlated stochastic knapsack problem. For this problem, Gupta et al. [22] gave a $1/8$ approximation algorithm for adaptive policies based on LP relaxation. The approximation ratio was improved to $1/(2 + \epsilon)$ by Ma [26], via a different time indexed LP formulation and a more sophisticated rounding scheme. Fukunaga et al. [16] extends the $1/8$ approximation algorithm, and achieve a $(1 - 1/\sqrt[4]{\epsilon})/2$ approximation for a case with submodular target function. This is achieved via a combination of the stochastic continuous greedy algorithm [2] (for getting a fractional solution), and the contention resolution scheme [13] (for rounding). A natural idea for improvement is to take ingredients from the $1/(2 + \epsilon)$ algorithm by Ma [26]. While the LP can be easily adapted, its rounding exhibits complicated dependencies that can be hard to analyze. We also have no luck with a direct application of the *contention resolution scheme* [8, 12, 13], a powerful technique in submodular optimization. The difficulties come in two folds. First, the original scheme is based on FKG inequality, which requires an *independently* rounded solution (possibly invalid) to start with. Any attempt to enforce our partition matroid at this step will break the whole scheme. This invalid solution is later fixed by ignoring some items from the rounded solution, and we need a way to impose the additional partition matroid constraint. Second, the ignoring step needs a critical “monotone”

⁴ For example, suppose we are left with a remaining budget of 20 at some time, and all items have a 0.001 probability of size 21. What this assumption suggests is that none of the items are allowed to be selected.

property. At a high level, the property says that the more items you choose, the lower the probability every other items will be selected (See Section 5.1 for a rigorous definition). While this may seem trivially true for any reasonable algorithm, it is not. In particular, it does not hold for Ma’s algorithm [26], due to its complicated dependencies. The first difficulty is not hard: if we happen to pick two items from the same partition, we just throw the later one out. Unfortunately, this makes the second obstacle even harder. The second obstacle is overcome by designing a brand-new rounding scheme which allows the direct analysis on the correlated probability of events. In order to achieve the aforementioned monotone property, we insert phantom items to block some “time slots” even when no item is there to conflict with. Such phantom items may be of independent interest for other applications of the contention resolution scheme. This alternate way of achieving monotonicity simultaneously free our analysis from one assumption mentioned in Section 1.3, which was needed in Fukunaga et al. [16] in their proof of the monotone property. A factor of $(1 - 1/\sqrt{e})$ is lost for the continuous optimization part, and another factor of 2 is lost for rounding, leading to our $(1 - 1/\sqrt{e})/2 \simeq 0.1967$ approximation algorithm.

2 Other Related Works

Stochastic Knapsack Problem. The stochastic version of the knapsack problem has long been studied. Kleinberg et al. [24], and Goel and Indyk [19] consider the stochastic version to maximize profit that will overflow the budget with probability at most p . However, they assume deterministic profits and special size distributions. Dean et al. [11] relax the limit on size and allow arbitrary distributions for item sizes. They investigate the gap between non-adaptive policies (the order of items to insert is fixed) and adaptive policies (allowed to make dynamic decision based on the realized size of items) and give a polynomial-time non-adaptive algorithm that approximates the optimal adaptive policy within a factor of $1/4$ in expectation. They also give an adaptive policy that approximates within a factor of $1/(3 + \epsilon)$ for any constant $\epsilon > 0$. Bhalgat et al. [4] improves on this and give a bi-criteria $(1 - \epsilon)$ algorithm by relaxing the budget by $(1 + \epsilon)$. Dean et al. [10] show that if correlation between size and reward is allowed, the problem would be PSPACE-hard. Gupta et al. [22] considered the case where the size and reward of an item can be arbitrarily correlated, and give an $1/8$ approximation. Li and Yuan [25] improved on this and get a $1/(2 + \epsilon)$ approximation with correlations and cancellation when ϵ fraction of extra space is allowed. This was further improved by Ma [26], who gets the same approximation ratio but without the budget augmentation requirement.

Submodular Maximization. Nemhauser et al. [27] studied the problem of maximizing a monotone submodular function subject to a cardinality constraint and gave the standard greedy $(1 - 1/e)$ -approximation algorithm. For the case with a matroid constraint, Fisher et al. [14] showed that the standard greedy algorithm gives a $1/2$ -approximation. This was improved to $(1 - 1/e)$ by Calinescu et al. [6], via the *continuous greedy algorithm*, which was originally developed by Calinescu et al. [5] for the submodular welfare problem. In this algorithm, the target function is relaxed via an exponential multilinear-extension. Though exponential, this version can be approximately solved to arbitrary precision in polynomial time. The fractional solution is then rounded via pipage rounding [5, 6, 28] or other rounding schemes [7]. In order to generalize the problem for other constraints and non-monotone submodular functions, a general rounding framework *contention resolution scheme* was proposed [6, 12, 13]. In this framework, the rounding step happens in two phases, an

independent rounding phase followed by a pruning phase, where the second phase ensures an upper bound on the probability that an element is pruned. One line of stochastic submodular optimization [3] assumes items have stochastic states, and would like to maximize a monotone submodular function on the stochastic states, under constraints on the set of chosen items. In other words, the constraints only depend on the selection of items, but not on the stochastic states of them. This is a generalization of the stochastic knapsack problem where the size of items are deterministic. Various settings of this problem are investigated by a series of follow-up works [20, 9, 15, 17, 18, 21, 31]. Asadpour and Nazerzadeh [2] considers the maximization of a monotone lattice-submodular function. In this problem, each selected item has a stochastic state (a non-negative *real* number). The target function accepts a vector of such numbers, and satisfies lattice-submodularity (defined in Section 3). In their problem, only the states are stochastic, while the matroid constraint is on the set of selected items. Fukunaga et al. [16] pushed one step further and allowed the constraints to be dependent on the state of items, but limited the set of states to be non-negative integers.

3 Preliminary

We start with some notations. The description of the spot scheduling problem [29] can be found in the full version [30], together with its modeling and reduction to the problem we consider. In Section 3.1, we explain how we reduce and manage to eliminate one critical assumption in the previous work by Fukunaga et al. [16].

Given two d dimensional vectors $u, v \in [n]^d$, we write $u \leq v$ if the inequality holds coordinate wise, i.e. $\forall i \in [d], u(i) \leq v(i)$. Similarly, $u \vee v$ and $u \wedge v$ are defined coordinate wise: $(u \vee v)(i) = \max\{u(i), v(i)\}$, $(u \wedge v)(i) = \min\{u(i), v(i)\}$. Consider a base set $[n]$, a matroid is defined to be an independent set $\mathcal{I} \subseteq 2^n$. This independent set needs to contain \emptyset , and if $A \in \mathcal{I}$, so is every $A' \subseteq A$. Furthermore, if $A, B \in \mathcal{I}$ and $|A| > |B|$, then there exists an element $x \in A \setminus B$ such that $B \cup x$ is in \mathcal{I} . Particularly, for a *partition matroid* $\{\mathcal{I}_k\}_{k \in [K]}$ where $\mathcal{I}_i \cap \mathcal{I}_j = \emptyset, \forall i \neq j$, its independent set \mathcal{I} is $\{S | \forall k, S \cap \mathcal{I}_k \leq 1\}$.

A function $f : 2^d \rightarrow \mathbb{R}$ is *submodular* if for every $A, B \subseteq [d]$, $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. An equivalent definition is that for every $A \subseteq B \subseteq [d]$ and $e \in [d]$, $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$. This definition is generalized to a domain of $[n]^d$, where function $f : [n]^d \rightarrow \mathbb{R}^+$ is called *lattice-submodular* if $f(u) + f(v) \geq f(u \wedge v) + f(u \vee v)$ holds for all $u, v \in [n]^d$. Note that the *lattice-submodularity* does not imply the property called *DR-submodularity*, which is the diminishing marginal returns along the direction of χ_i for each $i \in I$, where $\chi_i \in \{0, 1\}^n$, and only the i -th coordinate is 1. That is, $f(u + \chi_i) - f(u) \geq f(v + \chi_i) - f(v)$ does not necessarily hold for all $u, v \in [n]^d$ such that $u \leq v$ and $i \in [d]$ even if f is *lattice-submodular*. Function $f : [n]^d \rightarrow \mathbb{R}^+$ is called *monotone* if $f(u) \leq f(v)$ for all $u \leq v$.

3.1 Reduction and Eliminating an Assumption

In order to eliminate the second assumption mentioned in Section 1.3, we introduce the notion of a “size cap”. For each item i and a size cap b , we define an item (i, b) , where $p_{(i,b)}(s) = \pi_i(s)$ when $s < b$; $p_{(i,b)}(s) = \sum_{s' \geq s} \pi_i(s')$ when $s = b$; and 0 otherwise. The new reward function is exactly $R_{(i,b)}(\cdot)$.

We will be using a time-indexed LP formulation following Ma [26]. Instead of making a decision at each time step, we do it at each remaining size level. When there is enough room, we take item i itself into consideration. If the remaining size b is small, we are not able to get more reward for an item than when it has a size of b . Therefore, instead of trying to include

the original item i , we include item (i, b) , which is item i with size cap b . Obviously, we can include each item at most once. To achieve this, we impose a partition matroid $\{\mathcal{I}_i\}_{i \in [K]}$ on the items, where $\mathcal{I}_i = \{(i, b) | \forall b\}$. For the remainder of this paper, we view each (i, b) as an item, and the conflict between them is captured by the partition matroid.

4 Continuous Optimization Phase

Like most submodular maximization problems, our algorithm consists of two phases, a continuous optimization phase and a rounding phase. In this section, we describe the former.

4.1 Target Function

Given a lattice-submodular function $f : [M]^n \rightarrow \mathbb{R}^+$ and a distribution q_S of elements in set $S \subseteq [n]$, we define a set-submodular function $\bar{f} : 2^n \rightarrow \mathbb{R}_+$, where $\bar{f}(S) := \mathbb{E}_{r \sim q_S}[f(r)]$. This \bar{f} is guaranteed to be a monotone set-submodular function (See proof in [2]). Suppose the final selected (random) set is S , the value we are interested in would be $\mathbb{E}[\bar{f}(S)]$. Let \bar{x} be a vector, where $\bar{x}(i)$ denotes the probability that item i is in S . Using the well-established multi-linear extension, we define $\bar{F} : 2^n \rightarrow \mathbb{R}^+$, where $\bar{F}(\bar{x}) = \sum_{S \subseteq [n]} \prod_{i \in S} \bar{x}_i \prod_{i' \notin S} (1 - \bar{x}_{i'}) \bar{f}(S)$. This is the target function we are maximizing. Evaluating the function \bar{F} can take exponential time, but it can be approximated within a multiplicative factor of $(1 + \epsilon)$ for any constant $\epsilon > 0$. For simplicity, we assume $\bar{F}(\bar{x})$ can be evaluated exactly in this paper, which is standard in the literature (e.g. see [6]).

4.2 Stochastic Knapsack Exponential Constraints

The exponential and polynomial constraints on \bar{x} are adapted from Ma [26]. A group of exponential sized constraints describes the problem exactly. They are then relaxed to have a polynomial size, losing a factor of 2. For ease of notation, we follow Ma [26] and view an stochastic item i as an equivalent Markovian bandit, a special one that can force us to keep pulling it for a certain period of time. We use state $u_i(k, s)$ to indicate that arm i has been pulled k times, and the corresponding item has size s . From its initial state ρ_i , a single pull would decide the size s of this job, and move to state $u_i(1, s)$ respectively. We are then forced to keep pulling this arm (we will be using arm and item interchangeably) for the next $s - 1$ steps, and the last of such pulls moves us to its termination state \emptyset_i , and we can pull a new arm. Denote the probability of moving from state u to state v with $p_{u,v}$. After the first pull of item i , it moves to state $u_i(1, s)$ (having size s) with probability $p_{\rho_i, u_i(1, s)} = p_i(s)$. Therefore, if $k < s$, a pull will transit it to state $u_i(k + 1, s)$ with probability $p_{u_i(k, s), u_i(k+1, s)} = 1$. Otherwise, when $k = s$, transit to state \emptyset_i with probability $p_{u_i(k, s), \emptyset_i} = 1$, and we are allowed to pull a new arm.

Let π be a vector representing a joint state/node, where π_i denotes the state on item i . Let $S_i = \{u_i(*, *)\} \cup \{\rho_i, \emptyset_i\}$ for all $i \in [n]$, the set of all states for arm i , and $\tilde{\mathcal{S}} = S_1 \times \cdots \times S_n$, the set of all possible (maybe invalid) joint states. Let $\mathcal{S}' = \{\pi \in \tilde{\mathcal{S}} | \exists i \neq j, \pi_i \notin \{\rho_i, \emptyset_i\}, \pi_j \notin \{\rho_j, \emptyset_j\}\}$, the set of states where at least two arms are in the middle of processing at the same time, and $\mathcal{S}'' = \{\pi \in \tilde{\mathcal{S}} | \pi_i \neq \rho_i \text{ and } \pi_j \neq \rho_j, i, j \in \mathcal{I}_k \text{ for some } k\}$, the set of states where some conflicting arms (due to the partition matroid) have been started. Define $\mathcal{S} := \tilde{\mathcal{S}} \setminus (\mathcal{S}' \cup \mathcal{S}'')$, which is the set of all valid states. Let $I(\pi) = \{i | \pi_i \neq \emptyset_i\}$, the set of arms that could be played from state π . Let π^u denote the joint node where the component corresponding to u is replaced by u (note u can correspond to only one component). Let $y_{\pi, t}$

be the probability that we are at state π at time t , and $z_{\pi,i,t}$ the probability that we pulled arm i at time t , when the current state was π . Recall B is the total budget, we have the following basic constraints.

$$\sum_{i \in I(\pi)} z_{\pi,i,t} \leq y_{\pi,t} \quad \pi \in \mathcal{S}, t \in [B] \quad (1)$$

$$z_{\pi,i,t} = y_{\pi,t} \quad \pi \in \mathcal{S}, i : \pi_i \in S_i \setminus \{\rho_i, \emptyset_i\}, t \in [B] \quad (2)$$

$$z_{\pi,i,t} \geq 0 \quad \pi \in \mathcal{S}, i \in [n], t \in [B] \quad (3)$$

Let $\mathcal{A}_i = \{\pi \in \mathcal{S} : \pi_i \notin \{\rho_i, \emptyset_i\}\}$, the joint node when arm i is in the middle of processing. Note \mathcal{A}_i and \mathcal{A}_j are disjoint for $i \neq j$. We call arm i the *active* arm. Let $\mathcal{A} = \bigcup_{i \in [n]} \mathcal{A}_i$, the set of all states where some arm is *active*. For a state $\pi \in \mathcal{S}$, let $\mathcal{P}(\pi)$ denote the subset of \mathcal{S} that would transit to π with no play, which could happen when some arms turned inactive automatically: if $\pi \notin \mathcal{A}$, then $\mathcal{P}(\pi) = \{\pi\} \cup (\bigcup_{i \notin I(\pi)} \{\pi^u | u \in S_i \setminus \{\rho_i\}\})$; if $\pi \in \mathcal{A}$, then $\mathcal{P}(\pi) = \emptyset$. Suppose u corresponds to coordinate i , define $\text{Par}(u) = \{v \in S_i : p_{v,u} > 0\}$, the nodes that have a positive probability of transitioning to u . Then y -variables are updated as follows:

$$y_{(\rho_1, \dots, \rho_n), 0} = 1 \quad (4)$$

$$y_{\pi, 0} = 0, \quad \pi \in \mathcal{S} \setminus \{(\rho_1, \dots, \rho_n)\} \quad (5)$$

$$y_{\pi, t} = \sum_{\pi' \in \mathcal{P}(\pi)} \left(y_{\pi', t-1} - \sum_{i \in I(\pi')} z_{\pi', i, t-1} \right) \quad t > 0, \pi \in \mathcal{S} \setminus \mathcal{A} \quad (6)$$

$$y_{\pi, t} = \sum_{\rho_i \in \text{Par}(\pi_i)} \left(\sum_{\pi' \in \mathcal{P}(\pi^{\rho_i})} z_{\pi', i, t-1} \right) \cdot p_{\rho_i, \pi_i}, \quad t > 0, i \in [n], \pi \in \mathcal{A}_i, \pi_i \in \{u_i(1, *)\} \quad (7)$$

$$y_{\pi, t} = \sum_{u \in \text{Par}(\pi_i)} z_{\pi^u, i, t-1} \cdot p_{u, \pi_i}, \quad t > 0, i \in [n], \pi \in \mathcal{A}_i, \pi_i \notin \{u_i(1, *)\} \quad (8)$$

Equation (6) updates $y_{\pi, t}$ for $\pi \notin \mathcal{A}$, i.e. joint nodes with no active arms. Such a joint node π can only come from a no-play from a joint node in $\mathcal{P}(\pi)$. Equations (7) and (8) update $y_{\pi, t}$ for $\pi \in \mathcal{A}$. To get to the joint node π , we must have played arm i in previous step(s). In Equation (7), we consider the case if π_i is one of $u_i(1, *)$. We were at ρ_i right before, so it is possible that in the last step, we switched to π^{ρ_i} from some joint node in $\mathcal{P}(\pi^{\rho_i})$ without playing an arm. In Equation (8), we consider other cases, in which arm i was played at time $t-1$. These equations guarantee that at each time step, $y_{*, t}$ form a distribution, i.e. $\sum_{\pi \in \mathcal{S}} y_{\pi, t} = 1$. Combining this with Equation (1), we get $\sum_{\pi \in \mathcal{S}} \sum_{i \in I(\pi)} z_{\pi, i, t} \leq 1, \forall t \in [B]$. Equations (1)–(8) form the exponential constraints. We also need to relate these constraints with \bar{x} (recall $\bar{x}(i)$ is the probability that item i is included): $\bar{x}(i) = \sum_t \sum_{u \in S_i} \sum_{\pi \in \mathcal{S} : \pi_i = u} z_{\pi, i, t}$, which is the last missing piece for our exponential program, denoted as ExpP.

4.3 Stochastic Knapsack Polynomial Constraints

Obviously, we cannot solve this exponential program directly in polynomial time. In order to solve it, we relax the exponential program by disassemble the joint distribution of items. Let $s_{u, t}$ be the probability that arm i is on node u at the beginning of time t . Let $x_{u, t}$ be the probability that we pull an arm on node u at time t . Suppose $\mathcal{S} = \bigcup_i S_i$, we have the following constraints between $x_{u, t}$ and $s_{u, t}$.

$$x_{u, t} \leq s_{u, t} \quad u \in \mathcal{S}, t \in [B] \quad (9)$$

$$x_{u, t} = s_{u, t} \quad u \in \bigcup_{i \in [n]} S_i \setminus \{\rho_i, \emptyset_i\}, t \in [B] \quad (10)$$

$$x_{u, t} \geq 0 \quad u \in \mathcal{S}, t \in [B] \quad (11)$$

$$\sum_{u \in \mathcal{S}} x_{u, t} \leq 1 \quad t \in [B] \quad (12)$$

We also need constraints (13) for the partition matroid of arms (recall \mathcal{I}_k is a partition), and the state transition constraints (14)–(16).

$$\sum_{i \in \mathcal{I}_k} s_{\rho_i, 0} \leq 1, \quad \forall \mathcal{I}_k \qquad s_{\rho_i, 0} \geq 0, \quad i \in [n] \qquad (13)$$

$$s_{u, 0} = 0 \qquad u \in \mathcal{S} \setminus \{\rho_1, \dots, \rho_n\} \qquad (14)$$

$$s_{\rho_i, t} = s_{\rho_i, t-1} - x_{\rho_i, t-1} \qquad t > 0, i \in [n] \qquad (15)$$

$$s_{u, t} = \sum_{v \in \text{Par}(u)} x_{v, t-1} \cdot p_{v, u} \qquad t > 0, u \in \mathcal{S} \setminus \{\rho_1, \dots, \rho_n\} \qquad (16)$$

Relating these constraints with \bar{x} : $\bar{x}(i) = \sum_t \sum_{u \in \mathcal{S}_i} x_{u, t}$, we get the polynomial program PolyP. For any program $P \in \{\text{PolyP}, \text{ExpP}\}$, let OPT_P denote its optimal value.

4.4 Relating between the Exponential and the Polynomial Constraints

This was given in Ma [26], and we re-state for completeness without proof. The direction from ExpP to PolyP is trivial.

► **Theorem 1** (reformation of Lemma 2.3 from Ma [26]). *Given a feasible solution $\{z_{\pi, i, t}\}, \{y_{\pi, t}\}$ to ExpP, we can construct a solution to PolyP with the same objective value by setting $x_{u, t} = \sum_{\pi \in \mathcal{S}: \pi_i = u} z_{\pi, i, t}$, $s_{u, t} = \sum_{\pi \in \mathcal{S}: \pi_i = u} y_{\pi, t}$ for all $i \in [n]$, $u \in [0, 1]$, $t \in B$. Thus, the feasible region of PolyP is a projection of that of ExpP onto a subspace and $\text{OPT}_{\text{ExpP}} \leq \text{OPT}_{\text{PolyP}}$.*

For the other direction, we construct a solution $\{z_{\pi, i, t}, y_{\pi, t}\}$ of ExpP from a solution $\{x_{u, t}, s_{u, t}\}$ of PolyP, which obtains half its objective value. It will satisfy

$$\sum_{\pi \in \mathcal{S}: \pi_i = u} z_{\pi, i, t} = \frac{x_{u, t}}{2} \quad i \in [n], u \in \mathcal{S}_i, t \in [B].$$

We define specific $\{z_{\pi, i, t}, y_{\pi, t}\}$ over B iterations. On iteration t :

- Compute $y_{\pi, t}$ for all $\pi \in \mathcal{S}$.
- Define $\tilde{y}_{\pi, t} = y_{\pi, t}$ if $\pi \notin \mathcal{A}$, and $\tilde{y}_{\pi, t} = y_{\pi, t} - \sum_{a \in A} z_{\pi, i, t}$ if $\pi \in \mathcal{A}_i$ for some $i \in [n]$ (if $\pi \in \mathcal{A}_i$, then $\{z_{\pi, i, t} : a \in A\}$ is already set in a previous iteration).
- For all $i \in [n]$, define $f_{i, t} = \sum_{\pi \in \mathcal{S}: \pi_i = \rho_i} \tilde{y}_{\pi, t}$.
- For all $i \in [n]$, $\pi \in \mathcal{S}$ such that $\pi_i = \rho_i$, and $a \in A$, set $z_{\pi, i, t}^a = \tilde{y}_{\pi, t} \cdot \frac{1}{2} \cdot \frac{x_{\rho_i, t}}{f_{i, t}}$.
- For all $i \in [n]$, $\pi \in \mathcal{S}$ such that $\pi_i = \rho_i$ and $\pi_j \in \{\rho_j, \phi_j\}$ for $j \neq i$, define $g_{\pi, i, t} = \sum_{\pi' \in \mathcal{P}(\pi)} z_{\pi', i, t}$.
- For all $i \in [n]$, $u \in \mathcal{S}_i \setminus \{\rho_i\}$, $\pi \in \mathcal{S}$ such that $\pi_i = u$, and $a \in A$, set $z_{\pi, i, t + \text{depth}(u)}^a = g_{\pi, i, t} \cdot (x_{u, t + \text{depth}(u)}^a) / x_{\rho_i, t}$.

4.5 Solve the Continuous Optimization Problem

In order to solve PolyP, we follow Fukunaga et al. [16] and use the Stochastic Continuous Greedy algorithm. This algorithm maximizes the multi-linear extension G of a monotone set-submodular function g over a solvable downward-closed polytope. A polytope $\mathcal{P} \subseteq [0, 1]^{\mathcal{N}}$ is considered *solvable* if we can find an algorithm to optimize linear functions over it, and downward-closed if $x \in \mathcal{P}$ and $0 \leq y \leq x$ imply $y \in \mathcal{P}$. In our case, \mathcal{P} is solvable due to its linearity, and that solving a linear program falls in polynomial time. Note \mathcal{P} is down-monotone. The algorithm involves a controlling parameter called *stopping time*. For a stopping time $0 < b \leq 1$, the algorithm outputs a solution x such that $x/b \in \mathcal{P}$, while $G(x) \geq (1 - e^{-b} - O(n^3 \delta)) \max_{y \in \mathcal{Q}} G(y)$, where n is the size of the set over which g is defined and δ is the step size used in the algorithm. Here \mathcal{P} is assumed to include the characteristic vector of every singleton set.

► **Theorem 2** (reformation of Theorem 3 from Fukunaga et al. [16]). *If the stochastic continuous greedy algorithm with stopping time $b = 1/2 \in (0, 1]$ and step size $\delta = o(|I|^{-3})$ is applied to program PolyP , then the algorithm outputs a solution $x \in b\mathcal{P}$ such that $\bar{F}(\bar{x}) \geq (1 - e^{-b} - o(1))f_{\text{avg}}(\pi^*) \simeq 0.3935f_{\text{avg}}(\pi^*)$ for any adaptive policy π^* .*

5 Rounding Phase

Now that we have a fractional solution x , we proceed to round it to an integral policy (notice the fractional solution has already been scaled by a factor of 2). We need a variant of the contention resolution scheme that was introduced as a general framework for designing rounding algorithms that maximizes expected submodular functions ([8, 12, 13]). The variant is an extension from a set submodular function to a lattice-submodular function, first introduced in Fukunaga et al. [16]. We include its definition here for self-containment.

5.1 Contention Resolution Scheme

A contention resolution scheme (CRS) accepts a *pairwise independently* rounded solution which may violate some constraints, and fixes it without losing too much on expectation. Let $f : [B]^n \rightarrow \mathbb{R}_+$ be a monotone lattice-submodular function and the probability distribution $q_i : [B] \rightarrow [0, 1]$ on $[B]$ be given for each $i \in \{1, \dots, n\}$. We write $v \sim q$ if $v \in [B]^n$ is a random vector such that, for each $i \in \{1, \dots, n\}$, the corresponding component $v(i)$ is determined independently as $j \in [B]$ with probability $q_i(j)$. This is the independently rounded solution we feed into a CRS. Let $\mathcal{F} \subseteq [B]^n$ be a downward-closed subset of $[B]^n$, and let $\alpha \in [0, 1]$. We have the following definition for a α -CRS, its monotonicity, and one key property.

► **Definition 3** (α -Contention Resolution Scheme (α -CRS)). *A mapping $\psi : [B]^n \rightarrow \mathcal{F}$ is an α -CRS with respect to q if it satisfies:*

1. $\psi(v)(i) \in \{v(i), 0\}$ for each $i \in [n]$;
2. if $v \sim q$, then $\Pr[\psi(v)(i) = j | v(i) = j] \geq \alpha$ holds for all $i \in I$ and $j \in B$. The probability is based on randomness both in v and in ψ when ψ is randomized.

► **Definition 4** (monotone α -CRS). *An α -CRS ψ is considered monotone, if, for each $u, v \in [B]^n$ such that $u(i) = v(i)$ and $u \leq v$, $\Pr[\psi(u)(i) = u(i)] \geq \Pr[\psi(v)(i) = v(i)]$ holds. The probability is based only on the randomness of ψ .*

► **Lemma 5** (Theorem 4 from Fukunaga et al. [16]). *If ψ is a monotone α -CRS with respect to q , then $\mathbb{E}_{v \sim q}[f(\psi(v))] \geq \alpha \mathbb{E}_{v \in q}[f(v)]$.*

5.2 Rounding Algorithm

To fit in the contention resolution scheme, we need to first round everything independently. This means for each pair (i, t) , item i is scheduled at time t with probability $x_{\rho_i, t}$. Now we have a set $R' = \{(i, t)\}$ of proposed item time pairs. We sort the set according to t , and include the items one by one. Intuitively, for a pair (i, t) , we will only include item i if time t is available and does not invalidate the solution, i.e. each item is scheduled at most once, and at most one item from each partition. After including it in our solution, we get its realized size, and mark the corresponding time slots unavailable.

The main problem of this naive approach is that it does not exhibit monotonicity, which is a subtle but critical requirement for a CRS. To fix it, we schedule *phantom item* i even when we cannot fit it. We *simulate* its inclusion, and sample its size size_i should it be included. We

91:10 Correlated Stochastic Knapsack with a Submodular Objective

also mark those time slots corresponding to this *phantom item* unavailable, even when they are actually unoccupied. This seemingly wasteful step ensures that the rounding scheme is monotone. The final rounding algorithm is described in Algorithm 1.

■ **Algorithm 1** Rounding Algorithm.

```

1 foreach pair  $(i, t)$  do
2   | Sample  $(i, t)$  with probability  $x_{\rho_i, t}$ , and gets  $\emptyset$  otherwise;
3   | if not get  $\emptyset$  then  $I \leftarrow I \cup \{(i, t)\}$  ;
4 Sort  $I$  according to a non-decreasing ordering of  $t$ , break ties uniformly at random;
5  $C = 0, S = \emptyset$ , mark all times slots available;
6 for  $(i, t) \in I$  do
7   | if time slot  $t$  is available and item  $i$  does not violate constraints then
8   |   | Include item  $i$  and observe  $s_i$ ;
9   |   else
10  |   | Simulate including item  $i$ , and observe  $s_i$ ;
11  |   | Mark time slots from  $t$  to  $t + s_i$  unavailable;

```

The remaining of this section is devoted to proving the following theorem, which combined with Theorem 2 leads to our main result.

► **Theorem 6.** *Let π denote Algorithm 1, and x denote the solution we get from PolyP. Then $f_{avg}(\pi) \geq \bar{F}(\bar{x})/2$.*

To prove Theorem 6, we define two mappings $\sigma(\cdot)$ and $\omega(\cdot)$, where the first corresponds (roughly) to the step that maps x to I in Algorithm 1, and $\omega(\cdot)$ corresponds to the mapping (CRS) from set I to the final output. The mapping $\sigma(\bar{x})$ receives a real vector $\bar{x} \in [0, 1]^n$ and returns a random vector $v \in [B]^n$. From each partition \mathcal{I}_k , we pick at most one i , each $i \in \mathcal{I}_k$ is picked with probability $\bar{x}(i)$. If it is picked, the i -th component $v(i)$ independently takes value j with probability $p_i(j)$, and 0 otherwise, which happens with probability $1 - \sum_j p_i(j)$. This captures the construction of set I (only the item part, note $\Pr[\sigma(x)(i) > 0] = \Pr[\exists t, \text{s.t. } (i, t) \in I]$), together with the random outcome of the item. The mapping $\omega(\cdot)$ maps $v \in [B]^n$ to $w \in [B]^n$. To mimic Algorithm 1, we first assign time value $t(i)$ to each component $v(i)$, according to $x_{\rho_i, t}$. Based on $t(i)$, we form a precedence ordering \prec between i after random tie breaking (a random tie breaking is crucial). Then, we set $\omega(v)(i) = 0$ if there exists a component $j \prec i$ such that $t(j) \leq t(i) < t(j) + v(i)$, and $w(v)(i) = v(i)$ otherwise. We can observe that given input x , Algorithm 1 outputs exactly $\omega(\sigma(x))$ if the random realized sizes of items are the same. In order to prove Theorem 6, we need the following two lemmas. The first, whose proof in Fukunaga [16], corresponding to the independent rounding step, and the second corresponding to the CRS step.

► **Lemma 7.** $\mathbb{E}[f(\sigma(x))] \geq \bar{F}(\bar{x})$ holds for any $x \in P$.

► **Lemma 8.** ω is a 1/2-CRS with respect to \bar{x} .

► **Lemma 9.** The 1/2-CRS ω is monotone.

Lemma 7 is trivially true by the definition of $\bar{F}(\bar{x})$, which is the common starting point of contention resolution scheme. We first prove ω is a 1/2-CRS.

Proof of Lemma 8. Recall there are two properties needed for an α -CRS. The first property is obviously correct due to the definition of $\omega(\cdot)$. The second property needs to prove $\Pr[\omega(v)(i) = j | v(i) = j] \geq 1/2$. In the language of the rounding algorithm, let $\text{Drop}_{i,t}$ denotes the event (respect to the randomness in ω and v) that we drop the pair (i, t) . It is the same as proving

$$\Pr[\text{Drop}_{i,t} | \text{item } i \text{ is selected at time } t] \leq \frac{1}{2}.$$

Due to the way we round the solution, item i may be included more than once (at different times), and more than one item from the same partition may be included. Consider an item j at time t' (maybe the same as i) that could affect the pruning of item i at time t . Define $(j, t') \prec (i, t)$ if $t' < t$, or $t' = t$ and $j \prec i$. It is clear that (j, t') will affect (i, t) if and only if $(j, t') \prec (i, t)$. We slightly abuse notation, and let $\text{Drop}_{i,t}(j)$ denote the probability that the item j can causes the drop out of item i if a copy of it is scheduled at time t . Note this does not depend on whether item i is scheduled on t or not. We have:

► **Lemma 10.**

$$\text{Drop}_{i,t}(j) \leq \frac{1}{2} \sum_{u \in \{\emptyset_j\} \cup \{u_j(*,*)\}} x_{u,t} + \frac{1}{2} x_{\rho_j,t}.$$

Proof of Lemma 10. There are two cases and we bound the probability of dropping in each case.

Case 1. j belongs to the same partition as i ,

Case 2. j belongs to a different partition.

For the first case, the probability that it makes (i, t) invalid is

$$\begin{aligned} \text{Drop}_{i,t}(j) &\leq \frac{1}{2} (s_{\rho_j,0} - s_{\rho_j,t}) + \Pr[\text{item } j \text{ is considered before } i] \cdot \frac{1}{2} x_{\rho_j,t} \\ &\leq \frac{1}{2} \sum_{u \in \{\emptyset_j\} \cup \{u_j(*,*)\}} x_{u,t} + \frac{1}{2} x_{\rho_j,t}. \end{aligned}$$

The first term is the probability that at least one item j is scheduled before time t . Note this is actually an union bound due to our independent rounding. The second term is the probability that it is scheduled at time t , but will invalidate i since $j \prec i$. The second equality comes from the fact that if item j is scheduled some time before t , then it must be at some state at time t that is not the starting state ρ_j . In other words, either the end state \emptyset_j or some transient state $u_j(*,*)$.

For the second case, fix j , it can only drop i if it marked time slot t unavailable. The probability is

$$\begin{aligned} \text{Drop}_{i,t}(j) &\leq \frac{1}{2} \sum_{t'=1}^{t-1} x_{\rho_j,t'} \cdot \Pr[\text{size}_j \geq t - t'] + \Pr[\text{item } j \text{ is considered before } i] \cdot \frac{1}{2} x_{\rho_j,t} \\ &\leq \frac{1}{2} \sum_{t'=1}^{t-1} x_{\rho_j,t'} \cdot \Pr[\text{size}_j \geq t - t'] + \frac{1}{2} x_{\rho_j,t}. \end{aligned}$$

The first term is a summation of all the possible starting point of job j , times the probability that it will mark time slot t unavailable. Note this is also a union bound since there can be more than one copy of item j due to independent rounding. The second term is the probability that item j is also scheduled at time t , but is considered before i , i.e. $j \prec i$, which marks time slot t unavailable for i . We focus on the first term,

$$\begin{aligned}
& \sum_{t'=1}^{t-1} x_{\rho_j, t'} \cdot \Pr[\text{size}_j \geq t - t'] = \sum_{t'=1}^{t-1} \sum_{\tau=t-t'}^{B-t} x_{\rho_j, t'} \Pr[\text{size}_j = \tau] = \sum_{t'=1}^{t-1} \sum_{\tau=t-t'}^{B-t} x_{u_j(1, \tau), t'+1} \\
& = \sum_{t'=1}^{t-1} \sum_{\tau=t-t'}^{B-t} x_{u_j(t-t', \tau), t} \leq \sum_{u \in \{\emptyset_j\} \cup \{u_j(*, *)\}} x_{u, t}.
\end{aligned}$$

The last inequality holds because the index set of the summation on the left is a subset of that on the right. ◀

Therefore, the total probability that item i is blocked by any item is upper bounded by the union bound:

$$\begin{aligned}
\text{Drop}_{i, t} &= \sum_j \text{Drop}_{i, t}(j) \leq \frac{1}{2} \sum_j \sum_{u \in \{\emptyset_j\} \cup \{u_i(*, *)\}} x_{u, t} + \frac{1}{2} \sum_{j \in [n]} x_{\rho_j, t} \\
&\leq \frac{1}{2} \sum_{j \in [n]} \sum_{u \in \{\emptyset_j\} \cup \{u_i(*, *)\}} x_{u, t} + \frac{1}{2} \sum_{j \in [n]} x_{\rho_j, t} \leq \frac{1}{2} (1 - \sum_{j \in [n]} x_{\rho_j, t}) + \frac{1}{2} \sum_{j \in [n]} x_{\rho_j, t} = \frac{1}{2}. \quad \blacktriangleleft
\end{aligned}$$

Lastly, we show ω is monotone in the full version [30]. With everything ready, we can now prove Theorem 6, which combined with Theorem 2 leads to the main claim.

Proof of Theorem 6. The output r of Algorithm 1 satisfies $\mathbb{E}[f(r)] = \mathbb{E}[f(\omega(\sigma(x)))]$, and its feasibility is guaranteed by the algorithm. By Lemma 8 and Lemma 9, ω is a monotone $1/2$ -CRS with respect to q , where q is the probability defined in Lemma 8. Moreover, $\sigma(x) \sim q$ holds. By Lemma 5, $\mathbb{E}[f(\omega(\sigma(x)))] \geq \mathbb{E}[f(\sigma(x))]/2$. Using Lemma 7, we get $f_{\text{avg}}(\pi) = \mathbb{E}[f(r)] = \mathbb{E}[f(\omega(\sigma(x)))] \geq \mathbb{E}[f(\sigma(x))]/2 \geq \bar{F}(x)/2$. ◀

6 Conclusion

We consider the well studied correlated stochastic knapsack problem, generalizing its target function with submodularity to capture diminishing returns. An extra partition matroid constraint is added to generalize it and resolve an open question raised in a previous work to eliminate an assumption. We also make improvement on the approximation ratio. There is still a gap of 2 comparing to the variant with linear target function and we leave it as an open problem to close the gap.

References

- 1 Amr Ahmed, Choon Hui Teo, S.V.N. Vishwanathan, and Alex Smola. Fair and balanced: Learning to present news stories. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining, WSDM '12*, pages 333–342, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2124295.2124337.
- 2 Arash Asadpour and Hamid Nazerzadeh. Maximizing stochastic monotone submodular functions. *Management Science*, 62(8):2374–2391, 2016. doi:10.1287/mnsc.2015.2254.
- 3 Arash Asadpour, Hamid Nazerzadeh, and Amin Saberi. Stochastic submodular maximization. In *Proceedings of the 4th International Workshop on Internet and Network Economics, WINE '08*, pages 477–489, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-92185-1_53.
- 4 Anand Bhalgat, Ashish Goel, and Sanjeev Khanna. Improved approximation results for stochastic knapsack problems. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1647–1665. SIAM, 2011.

- 5 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. *Maximizing a Submodular Set Function Subject to a Matroid Constraint (Extended Abstract)*, pages 182–196. Integer Programming and Combinatorial Optimization. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-72792-7_15.
- 6 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011. doi:10.1137/080733991.
- 7 Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. Dependent randomized rounding for matroid polytopes and applications. *CoRR*, 2009. arXiv:0909.4348.
- 8 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. doi:10.1137/110839655.
- 9 Yuxin Chen and Andreas Krause. Near-optimal batch mode active learning and adaptive submodular optimization. In *Proceedings of the 30th International Conference on Machine Learning*, pages 160–168, 2013.
- 10 Brian C Dean, Michel X Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 395–404. SIAM, 2005.
- 11 Brian C Dean, Michel X Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of Operations Research*, 33(4):945–964, 2008.
- 12 Moran Feldman. *Maximization Problems with Submodular Objective Functions*. Ph.d. dissertation,, Technion - Israel Institute of Technology, 2013.
- 13 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*, October 2011. doi:10.1109/focs.2011.46.
- 14 M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. *An analysis of approximations for maximizing submodular set functions-II*, pages 73–87. Mathematical Programming Studies. Springer Berlin Heidelberg, 1978. doi:10.1007/bfb0121195.
- 15 Kaito Fujii and Hisashi Kashima. Budgeted stream-based active learning via adaptive submodular maximization. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL: <https://proceedings.neurips.cc/paper/2016/file/07cdfd23373b17c6b337251c22b7ea57-Paper.pdf>.
- 16 Takuro Fukunaga, Takuya Konishi, Sumio Fujita, and Ken ichi Kawarabayashi. Stochastic submodular maximization with performance-dependent item costs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:1485–1494, 2019. doi:10.1609/aaai.v33i01.33011485.
- 17 Victor Gabillon, Branislav Kveton, Zheng Wen, Brian Eriksson, and S. Muthukrishnan. Adaptive submodular maximization in bandit setting. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL: <https://proceedings.neurips.cc/paper/2013/file/f4573fc71c731d5c362f0d7860945b88-Paper.pdf>.
- 18 Victor Gabillon, Branislav Kveton, Zheng Wen, Brian Eriksson, and S. Muthukrishnan. Large-scale optimistic adaptive submodularity. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1816–1823, 2014.
- 19 Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 579–586. IEEE, 1999.
- 20 Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

- 21 Alkis Gotovos, Amin Karbasi, and Andreas Krause. Nonmonotone adaptive submodular maximization. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- 22 Anupam Gupta, Ravishankar Krishnaswamy, Marco Molinaro, and R Ravi. Approximation algorithms for correlated knapsacks and non-martingale bandits. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 827–836. IEEE, 2011.
- 23 Steven C. H. Hoi, Rong Jin, Jianke Zhu, and Michael R. Lyu. Batch mode active learning and its application to medical image classification. In *Proceedings of the 23rd International Conference on Machine Learning, ICML '06*, pages 417–424, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1143844.1143897.
- 24 Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30(1):191–217, 2000.
- 25 Jian Li and Wen Yuan. Stochastic combinatorial optimization via poisson approximation. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 971–980. ACM, 2013.
- 26 Will Ma. Improvements and generalizations of stochastic knapsack and markovian bandits approximation algorithms. *Mathematics of Operations Research*, 43(3):789–812, 2018. doi:10.1287/moor.2017.0884.
- 27 G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14(1):265–294, 1978. doi:10.1007/bf01588971.
- 28 Jan Vondrák. Symmetry and approximability of submodular maximization problems. In *2009 50th Annual Ieee Symposium on Foundations of Computer Science*, pages 251–270, October 2009. doi:10.1109/focs.2009.24.
- 29 Sheng Yang, Samir Khuller, Sunav Choudhary, Subrata Mitra, and Kanak Mahadik. Scheduling ML training on unreliable spot instances. In *2021 IEEE/ACM 14th International Conference on Utility and Cloud Computing (UCC '21) Companion*, 2021. doi:10.1145/3492323.3495594.
- 30 Sheng Yang, Samir Khuller, Sunav Chowdhary, Subrata Mitra, and Kanak Mahadik. Correlated stochastic knapsack with a submodular objective. *CoRR*, 2022. arXiv:2207.01551.
- 31 Baosheng Yu, Meng Fang, and Dacheng Tao. Linear submodular bandits with a knapsack constraint. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1380–1386, 2016.
- 32 Yisong Yue and Carlos Guestrin. Linear submodular bandits and their application to diversified retrieval. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL: <https://proceedings.neurips.cc/paper/2011/file/33ebd5b07dc7e407752fe773eed20635-Paper.pdf>.

Faster Algorithm for Unique $(k, 2)$ -CSP

Or Zamir

Institute for Advanced Study, Princeton, NJ, USA

Abstract

In a $(k, 2)$ -Constraint Satisfaction Problem we are given a set of arbitrary constraints on pairs of k -ary variables, and are asked to find an assignment of values to these variables such that all constraints are satisfied. The $(k, 2)$ -CSP problem generalizes problems like k -coloring and k -list-coloring. In the Unique $(k, 2)$ -CSP problem, we add the assumption that the input set of constraints has at most one satisfying assignment.

Beigel and Eppstein gave an algorithm for $(k, 2)$ -CSP running in time $O((0.4518k)^n)$ for $k > 3$ and $O(1.356^n)$ for $k = 3$, where n is the number of variables. Feder and Motwani improved upon the Beigel-Eppstein algorithm for $k \geq 11$. Hertli, Hurbain, Millius, Moser, Scheder and Szedlák improved these bounds for Unique $(k, 2)$ -CSP for every $k \geq 5$.

We improve the result of Hertli et al. and obtain better bounds for Unique $(k, 2)$ -CSP for $k \geq 5$. In particular, we improve the running time of Unique $(5, 2)$ -CSP from $O(2.254^n)$ to $O(2.232^n)$ and Unique $(6, 2)$ -CSP from $O(2.652^n)$ to $O(2.641^n)$.

Recently, Li and Scheder also published an improvement over the algorithm of Hertli et al. in the same regime as ours. Their improvement does not include quantitative bounds, we compare the works in the paper.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Algorithms, Constraint Satisfaction Problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.92

Funding Supported by NSF grant # CCF-1900460.

1 Introduction

The general Constraint Satisfaction Problem, in which we are asked to find an assignment to a set of variables that satisfies a list of arbitrary constraints, is NP-Complete. Furthermore, it is widely believed that a substantial improvement over the naive exhaustive search is unlikely for the general CSP problem and even for special cases of it like The *Boolean* Satisfiability Problem (SAT). Nevertheless, when the structure of the input is restricted in a certain manner, there are known improvements in the form of *moderately exponential* algorithms. These are algorithms that still have an exponential running time, yet achieve an exponential improvement over the exhaustive search bounds.

The study of moderately exponential algorithms for NP-Complete problems is extensive. In fact, exponential yet better-than-naive algorithms for NP-Complete problems were known for some problems, for example The Travelling Salesman Problem, long before the definition of NP. A survey of Woeginger [19] covers and refers to dozens of papers exploring such algorithms for many problems including satisfiability, graph coloring, knapsack, TSP, maximum independent sets and more. Subsequent review article of Fomin and Kaski [5] and book of Fomin and Kratsch [6] further cover the topic of exact exponential-time algorithms.

Two of the most notable problems for which the study of moderately exponential algorithms was fruitful are k -satisfiability (usually abbreviated as k -SAT) and graph coloring.

For satisfiability, the running time of the trivial algorithm enumerating over all possible assignments is $O^*(2^n)$. No algorithms solving SAT in time $O^*((2 - \epsilon)^n)$ for any $\epsilon > 0$ are known, and a popular conjecture called The *Strong* Exponential Time Hypothesis [3] states that no such algorithm exists. On the other hand, for every fixed k there exists a constant



© Or Zamir;

licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 92; pp. 92:1–92:13

Leibniz International Proceedings in Informatics



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

$\varepsilon_k > 0$ such that k -SAT (i.e., SAT on formulas in CNF form with at most k literals in every clause) can be solved in $O^*((2 - \varepsilon_k)^n)$ time. A result of this type was first published by Monien and Speckenmeyer in 1985 [13]. A long list of improvements for the values of ε_k were published since, including the celebrated 1998 PPSZ algorithm of Paturi, Pudlák, Saks and Zane [15] and the recent improvement over it by Hansen, Kaplan, Zamir and Zwick [7]. The PPSZ bound was originally obtained only for the case of input formulas with a unique satisfying assignment, its analysis was extended to the general case more than a decade later by Hertli [8].

For graph coloring, i.e., the problem of deciding whether a graph is k -colorable, the naive exhaustive search algorithm takes $O(k^n)$ time. Nevertheless, it is known that computing the chromatic (or coloring) number of a graph (i.e., the smallest k for which the graph is k -colorable) can be done in exponential time that does not depend on k . The first such result was an $O^*(3^n)$ algorithm of Lawler [10]. A long line of works followed until finally an algorithm computing the chromatic number in $O^*(2^n)$ time was devised by Björklund, Husfeldt and Koivisto in 2009 [2]. This is conjectured to be optimal. For $k \leq 6$ there are algorithms solving k -colorability exponentially faster than $O(2^n)$ [1][20]. It is currently not known even if 7-coloring can be solved exponentially faster than the general $O(2^n)$ bound of computing the coloring number. One of the biggest open problems in this field is whether k -coloring can be solved in $O^*((2 - \varepsilon_k)^n)$ time for every fixed k .

Both examples are special cases of the more general Constraint Satisfaction Problem. In an (a, b) -formula, we have n variables such that each of them can take a value in $[a] := \{1, \dots, a\}$ and a list of constraints such that each constraint may depend on at most b variables. Every such constraint can be equivalently replaced by a disjunction of at most a^b constraints of the form $(x_1 \neq c_1 \vee x_2 \neq c_2 \vee \dots \vee x_b \neq c_b)$ where x_1, \dots, x_b are (not necessarily distinct) variables and $c_1, \dots, c_b \in [a]$ are possible values. Thus, we can think of every (a, b) -formula as a list of constraints of that form. In the (a, b) -CSP problem we are given a (a, b) -formula and need to decide whether or not there is an assignment to the variables that satisfies all constraints. In the Unique (a, b) -CSP problem we add the assumption that if there is such an assignment it is unique. Note that k -SAT is the same as $(2, k)$ -CSP, and that k -coloring is a special case of $(k, 2)$ -CSP. We later elaborate on the close relation between the $(k, 2)$ -CSP and k -coloring problems.

In this paper we focus on obtaining better algorithms for Unique $(k, 2)$ -CSP.

1.1 Possible running time

Denote by $c_{a,b}$ the infimum of constants c such that (a, b) -CSP on formulas with n variables can be solved in $O((c + o(1))^n)$ time. Naively, $c_{a,b} \leq a$ as we can simply try all a^n possible assignments to the variables. A simple improvement comes from the use of *down-sampling*. Given a (a, b) -formula we may randomly restrict each variable to $a' < a$ uniformly chosen values. Each satisfying assignment is not ruled out by the restriction with probability $\left(\frac{a'}{a}\right)^n$. After this down-sampling step, we are left with a (a', b) -formula. Thus, for every $a' < a$ we have $c_{a,b} \leq \frac{a}{a'} \cdot c_{a',b}$. In particular, $c_{a,b} \leq \frac{a}{2} c_{2,b}$. As we know that k -SAT can be solved exponentially faster than $O(2^n)$ for every fixed k , we have that $c_{2,b} < 2$ and in particular the strict inequality $c_{a,b} < a$ holds for every a, b .

On the other hand, (a, b) -CSP is clearly NP-Complete for every $a > 1$ except of the special case of $(a, b) = (2, 2)$ (which is the polynomial 2-SAT). The Exponential Time Hypothesis [3] states that there exists some constant $c > 0$ such that 3-SAT takes $\Omega(2^{cn})$ time to solve. Traxler [18] showed that assuming the Exponential Time Hypothesis, there exists some $c' > 0$ such that $c_{k,2} > k^{c'}$. Namely, even for $b = 2$ the $(k, 2)$ -CSP problem becomes strictly more complex as k increases.

■ **Table 1** Comparisons of the exponent base in Unique $(k, 2)$ -CSP algorithms.

k	Downsampling+2SAT	PPZ FM [4]	BE [1]	PPSZ [9]	Our algorithm
3	1.5	1.818	<i>1.365</i>	1.434	-
4	2	2.214	<i>1.808</i>	1.849	-
5	2.5	2.606	2.259	<i>2.254</i>	2.232
6	3	2.994	2.711	<i>2.652</i>	2.641
7	3.5	3.381	3.163	<i>3.045</i>	3.042

1.2 $(k, 2)$ -CSP and k -Coloring

Consider the following hierarchy of three problems:

- **k -Coloring:** given a graph with n vertices, determine whether it is k -colorable.
- **k -List-Coloring:** given a graph with n vertices and a list of at most k allowed colors (from a possibly larger universe of colors) for each vertex, determine if there is a proper coloring of the graph such that each vertex is colored with one of the allowed colors in its list.
- **$(k, 2)$ -CSP:** given n variables that can admit values from $[k]$ and a list of arbitrary constraints involving one or two variables each, determine if there is an assignment of values to the variables that satisfies all constraints.

Each problem is a special case of the next one. Every instance of k -coloring is also an instance of k -list-coloring where all lists are simply $[k]$. Every instance of k -list-coloring is also an instance of $(k, 2)$ -CSP. Nevertheless, while k -coloring and k -list-coloring can both be solved in $O^*(2^n)$ time regardless of k [2], Traxler's reduction [18] shows that there is some constant k_0 such that for every $k > k_0$ we have $c_{k,2} \geq 2$. Let k_0 be the minimal such constant. It is currently known that $c_{4,2} < 2$ and thus $k_0 \geq 4$.

In [20] it was recently shown that if k -list-coloring can be solved in $O^*((2 - \varepsilon)^n)$ time for some $\varepsilon > 0$ then $(k + 2)$ -coloring can also be solved in $O^*((2 - \varepsilon')^n)$ for some $\varepsilon' > 0$. In particular, $(k_0 + 2)$ -coloring can be solved exponentially faster than $O(2^n)$. As $k_0 \geq 4$, this resulted in the first $O^*((2 - \varepsilon)^n)$ time algorithms for 5-coloring and 6-coloring. This gives a strong motivation for improving upper bounds for $(k, 2)$ -CSP, with the goal of improving the bound on k_0 . In particular, showing that $(5, 2)$ -CSP can be solved in $O^*((2 - \varepsilon)^n)$ time would result in the first $O^*((2 - \varepsilon)^n)$ time algorithm for 7-coloring.

1.3 Previous results

By the down-sampling argument we can reduce $(k, 2)$ -CSP to the polynomial $(2, 2)$ -CSP (i.e., 2-SAT) and get an expected running time of $O\left(\left(\frac{k}{2}\right)^n\right)$. Beigel and Eppstein [1] gave an algorithm for $(k, 2)$ -CSP running in time $O((0.4518k)^n)$ for $k > 3$ and $O(1.356^n)$ for $k = 3$. Feder and Motwani [4] give a $(k, 2)$ -CSP algorithm based on the k -SAT PPZ algorithm, which is the predecessor of the PPSZ one. They improve on the bound of Beigel and Eppstein only for $k \geq 11$. Hertli, Hurbain, Millius, Moser, Scheder and Szedlák [9] improved the bounds for Unique $(k, 2)$ -CSP for every $k \geq 5$. Several other works [17] [11] [16] focus on the case where $b > 2$.

1.4 Our contribution

We present an algorithm that improves on the result of Hertli et al. and obtain better bounds for Unique $(k, 2)$ -CSP for $k \geq 5$. In particular, we improve the running time of Unique $(5, 2)$ -CSP from $O(2.254^n)$ to $O(2.232^n)$ and Unique $(6, 2)$ -CSP from $O(2.652^n)$ to $O(2.641^n)$. Our result is compared to the previous ones in Table 1.

We obtain our result by combining the strengths of both PPSZ and the Beigel-Eppstein algorithms. Intuitively, we make the following insight regarding PPSZ-type algorithms. Throughout the run of the PPSZ algorithm, it slowly manipulates the CSP formula. For every $k' < k$ there is some time-point such that if we stop the algorithm at that point then the formula roughly looks like a $(k', 2)$ -CSP formula. Furthermore, the rest of the algorithm run would have looked similar to running PPSZ on a $(k', 2)$ -formula. Thus, for $k \geq 5$ we may stop the run of the PPSZ algorithm when the formula looks similar to a $(4, 2)$ -CSP or $(3, 2)$ -CSP formula, and then switch to using the Beigel-Eppstein algorithm which is faster than PPSZ for $k \leq 4$.

In Section 2 we give an extensive overview of the previous results we need to use. Then in Section 3 we introduce our algorithm and obtain the improved bounds. In Section 4 we sketch possible improvements to the analysis of Section 3 and show that even with a completely ideal analysis of our algorithm we would improve the bound for $k = 5$ from $O(2.232^n)$ only to $O(2.223^n)$. Thus, to prove that $k_0 \geq 5$, if true, additional algorithmic tools are necessary. In Section 5 we conclude our work, discuss more possible uses for the PPSZ-related observations, and present open problems.

1.5 Comparison with a recent work of Li and Scheder

Recently, Li and Scheder [12] also published an improvement for the PPSZ-type algorithm of Hertli et al. for Unique CSP. Their algorithm does not use the Beigel-Eppstein algorithm and just modifies the PPSZ-type algorithm itself in a different manner to ours. They prove that this modification gives an exponential improvement over the bounds of Hertli et al., but do not give a quantitative bound of this improvement. We could not find a way to compute such a bound from their paper, but suspect this improvement is very small.

Li and Scheder also observe that during the run of the PPSZ-type algorithm when the number of color choices of a variable is very low (in their algorithm, when it reaches 2), then there are better ways to settle the value of the variable than continuing the run of the PPSZ algorithm. In their case, they do so by randomly picking one of the two colors for the variable.

Our work can be seen as a refinement of this idea in two different ways. First, we cut off the PPSZ run with small color sets yet larger than two. Second, we resolve the remaining instance with a variant of the Beigel-Eppstein algorithm, which is much better than a random choice.

2 Relevant overview of previous work

In this section we give an overview of all previous results that are used in our algorithm. We repeat and refine some of the theorems used in these papers for their later use in Section 3.

2.1 The algorithm of Beigel and Eppstein

The algorithm of Beigel and Eppstein [1] solves a CSP by performing a series of local reductions that either reduce the number of variables in the CSP or the number of allowed values in some of the variables.

An example for such a local reduction that is of particular interest to us follows.

► **Lemma 1** (Lemma 2 of [1]). *Let (V, F) be a CSP in which each variable $x \in V$ has $k(x)$ allowed values. Let x be a variable with $k(x) = 2$, then there exists a set F' of additional constraints, each of size two, such that (V, F) is satisfiable if and only if $(V \setminus \{x\}, F \cup F')$ is satisfiable, with the same number of allowed values for each variable $y \neq x$.*

The result claimed in [1] is that their algorithm solves $(3, 2)$ -CSPs in time $O(1.3645^n)$ and $(k, 2)$ -CSPs for $k > 3$ in time $O((0.4518k)^n)$. Note that $1.3645 > 1.3554 = 0.4518 \cdot 3$. In fact, the result proved in their paper is slightly stronger than that. The following Theorem follows from [1].

► **Theorem 2** (Section 5 of [1]). *Let (V, F) be a CSP with $|V| = n_3 + n_4$ variables such that n_3 variables have three allowed values and n_4 variables have four allowed values, then we can solve it in time $O(1.3645^{n_3} \cdot 1.8072^{n_4})$.*

The claimed results follow from Theorem 2 immediately. For $(3, 2)$ -CSPs we simply have $n_3 = n$, $n_4 = 0$ and for $(k, 2)$ -CSPs with $k > 3$ we down-sample each variable to 4 out of its k possible values and then use the theorem with $n_3 = 0$, $n_4 = n$, with a total expected run-time of $O\left(\left(\frac{k}{4}\right)^n \cdot 1.8072^n\right) = O((0.4518k)^n)$. Nevertheless, for our use we need to fully use the power of Theorem 2 and we even slightly refine it with the following statement, from now on referred to as *the extended BE algorithm*.

► **Theorem 3.** *Denote by*

$$BE(i) := \begin{cases} 1 & \text{if } i \leq 2 \\ 1.3645 & \text{if } i = 3. \\ 0.4518 \cdot i & \text{if } i \geq 4 \end{cases}$$

Let (V, F) be a CSP in which each variable $x \in V$ has $k(x)$ allowed values. Let n_i be the number of variables x in V such that $k(x) = i$. Then, we can solve (V, F) in $O(\prod_i BE(i)^{n_i})$ expected time.

Proof. The n_1 variables with a single possible value can be ignored. The n_2 variables with two possible values can be eliminated using Lemma 1. For every $i > 4$ we use down-sampling to reduce the number of allowed values to four. We finally apply Theorem 2. ◀

2.2 The PPZ and PPSZ-type algorithms

In this section we present an overview of [4] and [9]. We state theorems of both papers and their variants that are useful for our analysis, and adapt some of their notation. Throughout the section we discuss only instances with a unique satisfying assignment.

► **Definition 4** (*D-implication*). *Let F be a $(k, 2)$ -CSP formula over a set V of variables, $x \in V$ be a variable, $c \in [k]$ be a possible value, α_0 a partial assignment, and $D \in \mathbb{N}$. We say that α_0 *D-implies* $x \neq c$ and write $\alpha_0 \models_D (x \neq c)$ if there is a subset of constraints $G \subseteq F$ of size $|G| \leq D$ such that $G \wedge \alpha_0$ implies $(x \neq c)$.*

By enumeration, we can check whether $\alpha_0 \models_D (x \neq c)$ in $O(|F|^D \cdot \text{poly}(n))$ time, which is polynomial in n, k if D is a constant and sub-exponential in n even if D is a slow-enough growing function of n . For the rest of the section we fix D .

► **Definition 5** (*Eligible values*). *Let F be a $(k, 2)$ -CSP formula over a set V of variables, α_0 a partial assignment, and $x \in V \setminus V(\alpha_0)$ a variable which value is not assigned in α_0 . We denote by*

$$\mathcal{A}(x, \alpha_0) := \{c \in [k] \mid \alpha_0 \not\models_D (x \neq c)\}$$

the set of all possible values for x that are not ruled out by D -implication from α_0 .

92:6 Faster Algorithm for Unique $(k, 2)$ -CSP

We can now describe the PPSZ algorithm (adapted from SAT [14] to CSP in [9]). Given a $(k, 2)$ -CSP F , we begin with $\alpha_0 = \emptyset$ the empty assignment and incrementally add variables to it, hoping to finish with a satisfying assignment. In particular, we choose a permutation π of the variables V uniformly at random, and then choose an assignment for the variables of V one-by-one according to the order of π . When we reach a variable x , we compute $\mathcal{A}(x, \alpha_0)$ in sub-exponential time, pick a uniformly random $c \sim U(\mathcal{A}(x, \alpha_0))$, and extend α_0 by setting $\alpha_0(x) = c$.

■ **Algorithm 1** The PPSZ algorithm.

```

Pick a uniform random permutation  $\pi$  of the set  $V$  of variables;
Set  $\alpha_0 = \emptyset$ ;
for  $x \in V$  in the order dictated by  $\pi$  do
    | Draw  $c \sim U(\mathcal{A}(x, \alpha_0))$  uniformly;
    | Set  $\alpha_0(x) := c$ ;
Return  $\alpha_0$ ;

```

Algorithm 1 runs in sub-exponential time and returns some assignment α_0 to all variables of V . It is clear that the probability of α_0 to satisfy F , assuming that F is satisfiable, is at least k^{-n} . We next prove that for formulas F with exactly one satisfying assignment α , the probability that Algorithm 1 produces the satisfying assignment $\alpha_0 = \alpha$ is in fact exponentially larger. For the rest of the section we assume that F has a unique satisfying assignment and denote it by α .

► **Definition 6** (Ultimately eligible values). *Let F be a $(k, 2)$ -CSP formula uniquely satisfied by α , π be a permutation of its variables V and $x \in V$ some variable.*

We let $V_{\pi, x} := \{y \in V \mid \pi(y) < \pi(x)\}$ be the set of all variables appearing before x in π , $\alpha_{\pi, x} := \alpha|_{V_{\pi, x}}$ be the partial assignment resulting by restricting α to $V_{\pi, x}$ and then we denote by $\mathcal{A}(x, \pi) := \mathcal{A}(x, \alpha_{\pi, x})$ the set of all possible values for x that are not ruled out by D -implication when we reach x in a PPSZ iteration with permutation π , given that all previous variables were set correctly.

We observe that Algorithm 1 returns α if and only if it draws the correct value for *every* variable. In particular, for a specific permutation π the probability of success is exactly $\prod_{x \in V} \frac{1}{|\mathcal{A}(x, \pi)|}$. For a random permutation then, the probability of success is

$$\mathbb{E}_{\pi} \left[\prod_{x \in V} \frac{1}{|\mathcal{A}(x, \pi)|} \right] \geq k^{-\sum_{x \in V} \mathbb{E}_{\pi} [\log_k |\mathcal{A}(x, \pi)|]}$$

where we use Jensen's inequality. In particular, it is enough to give an upper bound on $\mathbb{E}_{\pi} [\log_k |\mathcal{A}(x, \pi)|]$ that holds for every variable x .

2.3 The PPZ-type algorithm of Feder and Motwani

In the PPZ-type variant of Feder and Motwani [4], a simpler variant where $D = 1$ is presented and analysed. Namely, a possible value c for a variable x is ruled out if and only if a variable y appeared before x in the permutation and was assigned a value c' such that the constraint $(x \neq c \vee y \neq c')$ appears in the list of constraints. When we reach the variable x , we uniformly guess a value for it out of all values that are not ruled out in that manner.

► **Lemma 7.** *Let (V, F) be the sets of variables and constraints in a Unique $(k, 2)$ -CSP. Denote by φ the unique satisfying assignment of (V, F) . For every variable $x \in V$ and every value $\varphi(x) \neq c' \in [k]$ other than the value x is assigned in φ , there exists a variable $y = y_{x,c'} \in V \setminus \{x\}$ such that $(x \neq c' \vee y \neq \varphi(y)) \in F$.*

Proof. Assume by contradiction that there exists a variable x and a value $c' \neq \varphi(x)$ for which $(x \neq c' \vee y \neq \varphi(y)) \notin F$ for every variable y . Consider the assignment φ' such that $\varphi'(x) = c'$ and $\varphi'(y) = \varphi(y)$ for every $y \neq x$. It is a satisfying assignment as well, and $\varphi' \neq \varphi$ which contradicts the uniqueness assumption. ◀

Instead of uniformly drawing a permutation $\pi \sim S_{|V|}$ of the variables, we (equivalently) independently draw a *time* value $\pi(x) \sim U([0, 1])$ uniformly for every variable x , and let π be the permutation induced by the order of the time values $\pi(x)$ for $x \in V$.

We observe that if $\pi(y_{x,c'}) < \pi(x)$ then $c' \notin \mathcal{A}(x, \pi)$. In particular, if $\pi(x) = p \in [0, 1]$ then for every $c' \neq \varphi(x)$ with probability at least p we have that $c' \notin \mathcal{A}(x, \pi)$.

► **Lemma 8.** *For every variable x , we have*

$$\mathbb{E}_\pi [\log_k |\mathcal{A}(x, \pi)| \mid \pi(x) = p] \leq \sum_{i=0}^{k-1} \binom{k-1}{i} (1-p)^i p^{k-1-i} \log_k(1+i).$$

Proof. For the analysis, we may assume that we rule out values only by the constraints involving x and one of the variables $y_{x,c'}$ for $c' \neq \varphi(x)$. This holds since ruling out more variables can only decrease the size of $\mathcal{A}(x, \pi)$.

If the variables $y_{x,c'}$ are distinct for all $c' \neq \varphi(x)$, then the right hand side of the lemma's statement is exactly the expected size of $\mathcal{A}(x, \pi)$, conditioned on $\pi(x) = p$. This is simply the expectation of $\log_k(1+i)$ where $i \sim \text{Binomial}(k-1, 1-p)$ is a binomial random variable.

Generally, let $A_{c'}$ be the indicator for the event that $\pi(y_{x,c'}) > p$. We need to upper bound $\mathbb{E}[\log_k(1 + \sum_{c' \neq \varphi(x)} A_{c'})]$. Let $A'_{c'}$ be independent Bernoulli random variables with probability $(1-p)$ to be 1 and probability p to be 0. By concavity of the function $\log_k(1+z)$ and Jensen's inequality it follows that $\mathbb{E}[\log_k(1 + \sum_{c' \neq \varphi(x)} A_{c'})] \leq \mathbb{E}[\log_k(1 + \sum_{c' \neq \varphi(x)} A'_{c'})]$, which concludes our proof. The complete proof of the last statement appears in [9] as Lemma A.1. ◀

Denote by $S'_{k,2} := \int_0^1 \sum_{i=0}^{k-1} \binom{k-1}{i} (1-p)^i p^{k-1-i} \log_k(1+i) dp$. By Lemma 8, $\mathbb{E}[\log_k |\mathcal{A}(x, \pi)|] = \int_0^1 \mathbb{E}_\pi [\log_k |\mathcal{A}(x, \pi)| \mid \pi(x) = p] dp \leq S'_{k,2}$, this concludes the analysis of the Feder-Motwani PPZ-type algorithm.

► **Theorem 9 ([4]).** *The success probability of a PPZ iteration is at least $k^{-S'_{k,2}}$.*

2.4 The PPSZ-type algorithm of Hertli et al.

In the PPSZ algorithm analysed in [9] more involved D -implications are considered. In the analysis for $D = 1$, we noticed that for every variable x and every value $c' \neq \varphi(x)$ there exists some variable $y_{x,c'}$ such that if $\pi(y_{x,c'}) < \pi(x)$ then $c' \notin \mathcal{A}(x, \pi)$.

We say that a variable y is *decided* with respect to some partial assignment α_0 if $|\mathcal{A}(y, \alpha_0)| = 1$, i.e., if α_0 already D -implies the correct value of y in φ . The main observation is that if in time $p := \pi(x)$ the variable $y_{x,c'}$ is decided then $c' \notin \mathcal{A}(x, \pi)$. The variable $y_{x,c'}$ is necessarily decided if $\pi(y_{x,c'}) < p$ but can also be decided if it is yet to appear in the permutation. Thus, the probability of $y_{x,c'}$ being decided at time p is strictly larger than p .

We give an intuitive reasoning for the probability of a variable being decided. Denote by $q_k(p)$ the probability that a variable x is decided by time p . The variable x is decided by time p if $\pi(x) < p$ or alternatively if for every $c' \neq \varphi(x)$ the variable $y_{x,c'}$ is by itself decided at time p . In particular, $q_k(p)$ is a solution to the recurrence $q_k(p) = p + (1 - p)q_k(p)^{k-1}$. We thus denote by $q_k(p)$ the smallest non-negative real solution to that recurrence, it can be analytically computed for every k as it is simply a root of a polynomial.

This intuitive argument is of course not complete and lacks many technical details. Nevertheless, this statement does hold, and the following strengthening of Lemma 8 and Theorem 9 are proven in [9].

► **Lemma 10** (A.1 in [9]). *For every variable x , we have*

$$\mathbb{E}_\pi [\log_k |\mathcal{A}(x, \pi)| \mid \pi(x) = p] \leq \sum_{i=0}^{k-1} \binom{k-1}{i} (1 - q_k(p))^i q_k(p)^{k-1-i} \log_k(1+i).$$

Denote by $S_{k,2} := \int_0^1 \sum_{i=0}^{k-1} \binom{k-1}{i} (1 - q_k(p))^i q_k(p)^{k-1-i} \log_k(1+i) dp$.

► **Theorem 11** (Correctness of [9]). *Let F be a Unique $(k, 2)$ -CSP formula, then for every variable x it holds that $\mathbb{E}_\pi [\log_k |\mathcal{A}(x, \pi)|] \leq S_{k,2} + \varepsilon_D$, where ε_D is some error parameter that depends only on D and goes to 0 as D goes to infinity.*

3 Faster Unique $(k, 2)$ -CSP algorithm

On a very high-level, our algorithm combines Hertli et al.'s PPSZ (Section 2.2) with the BE algorithm (Section 2.1). We begin by illustrating our idea intuitively (initially ignoring some crucial technical details to be discussed later). Consider a run of the PPSZ algorithm, as described in Section 2.2. For the early variables in the permutation π , it is very likely that $|\mathcal{A}(x, \pi)| = k$, since α_0 assigns values to very few variables. On the other hand, for the last variables in the permutation, it is very likely that $|\mathcal{A}(x, \pi)| = 1$. It turns out that in any point throughout the run of a PPSZ iteration, the sizes $|\mathcal{A}(x, \alpha_0)|$ for the remaining variables $x \in V \setminus V(\alpha_0)$ are quite concentrated. Furthermore, after most of the variables have $|\mathcal{A}(x, \alpha_0)| \approx k' < k$ the remaining portion of the PPSZ iteration strongly resembles a PPSZ algorithm for $(k', 2)$ -CSP formulas. As we see in Table 1, for $k < 5$ PPSZ behaves worse on $(k, 2)$ -CSP formulas than the BE algorithm.

Thus, in our algorithm, we begin with an iteration of PPSZ but halt it somewhere in the middle of the permutation when the sizes $|\mathcal{A}(x, \alpha_0)|$ are concentrated in 1, 2, 3, 4. At that point, we use the extended BE algorithm shown in Section 2.1.

We set some parameter $t \in [0, 1]$ to be chosen later and consider the following algorithm.

■ **Algorithm 2** Our algorithm.

Pick a uniform random permutation π of the set V of variables;
 Denote by $\pi_{<t}$ the prefix of π of size $t|V|$ and by $V_{<t}$ the variables appearing in it;
 Set $\varphi = \emptyset$;
for $x \in V_{<t}$ *in the order dictated by $\pi_{<t}$* **do**
 | Draw $c \sim U(\mathcal{A}(x, \varphi))$ uniformly;
 | Set $\varphi(x) := c$;
 Run the extended BE algorithm on the remaining CSP F ;
 Return the solution φ ;

Consider an iteration of Algorithm 2. Denote by R_i the number of variables that appeared in $V_{<t}$ and had $|\mathcal{A}(x, \pi)| = i$. Denote by φ' the partial assignment constructed by time t . Let B_i be the number of variables that did not appear in $V_{<t}$ and had $|\mathcal{A}(x, \varphi')| = i$.

► **Lemma 12.** *The success probability of Algorithm 2 is $\prod_{i=1}^k i^{-R_i} \cdot \prod_{i=5}^k \left(\frac{4}{i}\right)^{B_i}$.*

Proof. The probability of all PPSZ assignments to be correct is $\prod_{i=1}^k i^{-R_i}$, as follows from Section 2.2. The probability of the random down-sampling to not rule out the correct assignments is $\prod_{i=5}^k \left(\frac{4}{i}\right)^{B_i}$. ◀

► **Lemma 13.** *The running time of Algorithm 2 is $O(1.3645^{B_3} \cdot 1.8072^{B_4+\dots+B_k})$.*

Proof. The running time of the (partial) PPSZ iteration is polynomial. The running time of the BE algorithm is $O(1.3645^{n_3} \cdot 1.8072^{n_4})$. ◀

Note that all R_i and B_i are fully determined by the choice of π . Thus, for a specific choice of π , if we repeatedly run Algorithm 2 with π we expect finding a solution after $\left(\prod_{i=1}^k i^{-R_i} \cdot \prod_{i=5}^k \left(\frac{4}{i}\right)^{B_i}\right)^{-1}$ iterations. In particular, after

$$\left(\prod_{i=1}^k i^{-R_i} \cdot \prod_{i=5}^k \left(\frac{4}{i}\right)^{B_i}\right)^{-1} \cdot O(1.3645^{B_3} \cdot 1.8072^{B_4+\dots+B_k}) = \prod_{i=1}^k i^{R_i} \cdot \prod_i BE(i)^{B_i}$$

computational steps. At this point, we would like to bound the expected running time when picking a random π with $\prod_{i=1}^k i^{\mathbb{E}[R_i]} \cdot \prod_i BE(i)^{\mathbb{E}[B_i]}$. Unfortunately, as we consider the running time and not a success probability (as in the PPSZ algorithm), we need an inequality of the opposite direction to Jensen's inequality. Fortunately, this inequality *essentially still holds* in this case.

► **Lemma 14** (Wrong direction Jensen's inequality is still kind-of right). *Let \mathcal{A} be an algorithm with expected running time 2^X conditioned on the value of a random variable X . There exists an algorithm \mathcal{A}' that successfully executes \mathcal{A} with probability at least 0.99 and has an expected running time of $O(2^{\mathbb{E}[X]} \cdot \mathbb{E}[X])$.*

Proof. We apply Markov's inequality twice. First, to observe that

$$\Pr(X > \mathbb{E}[X] + 1) \leq \frac{1}{1 + \frac{1}{\mathbb{E}[X]}} = 1 - \frac{1}{\mathbb{E}[X] + 1}.$$

Hence, if we run \mathcal{A} independently for $6(\mathbb{E}[X] + 1)$ times, then with probability at least $1 - e^{-6} > 1 - \frac{1}{200}$ at least one of these runs has $X \leq \mathbb{E}[X] + 1$. Second, conditioned on any value of X , with probability at least $1 - \frac{1}{200}$ algorithm \mathcal{A} finishes in less than $200 \cdot 2^X$ computational steps. Thus, by union bound, if we run algorithm \mathcal{A} for $6(\mathbb{E}[X] + 1)$ times, and terminate each run after $400 \cdot 2^{\mathbb{E}[X]}$ computational steps, then at least one run of \mathcal{A} finishes with probability at least 0.99. ◀

► **Corollary 15.** *We find a satisfying assignment with probability greater than 0.99 in time*

$$O^* \left(\prod_{i=1}^k i^{\mathbb{E}[R_i]} \cdot \prod_i BE(i)^{\mathbb{E}[B_i]} \right). \quad (\star)$$

92:10 Faster Algorithm for Unique $(k, 2)$ -CSP

We give a simpler analysis leading to slightly sub-optimal bounds. In Section 4 we sketch the possible improvements to the analysis presented here, and also present a clear limit to the improvements that can be achieved by this algorithm.

We slightly abuse notation by equating the numbers R_i, B_i with the sets of variables they are counting. Let x be a variable. For the analysis we can assume that the algorithm rules out values for x only due to the constraints involving x and some $y_{x,c'}$. This holds as ruling out more values can only improve the success probability of each iteration.

We first consider the case in which the variables $y_{x,c'}$ for every $c' \neq \varphi(x)$ are all distinct.

► **Lemma 16.** *For each variable x , we have*

$$\mathbb{E} \left[\sum_{i=1}^k Pr(x \in R_i) \cdot \log_k i \right] \leq \int_0^t \sum_{i=0}^{k-1} \binom{k-1}{i} (1 - q_k(p))^i q_k(p)^{k-1-i} \log_k(1+i) dp.$$

Proof. This follows immediately from Lemma 10. ◀

► **Lemma 17.** *Let x be a variable for which $y_{x,c'}$ are distinct for all $c' \neq \varphi(x)$. Then,*

$$\mathbb{E} \left[\sum_{i=3}^k Pr(x \in B_i) \cdot \log_k EB(i) \right] \leq (1-t) \cdot \sum_{i=3}^k \binom{k-1}{i} (1-t)^i t^{k-1-i} \cdot \log_k EB(i).$$

Proof. For simplicity, we analyse this part with a PPZ-type analysis (rather than PPSZ-type one), this is further discussed in Section 4. The probability that $x \notin V_{<t}$ is $(1-t)$, and the probability that exactly i out of the $(k-1)$ variables $y_{x,c'}$ do not appear in $V_{<t}$ is $\binom{k-1}{i} (1-t)^i t^{k-1-i}$. ◀

Denote by

$$\begin{aligned} \text{cost}(k, t) &:= \int_0^t \sum_{i=0}^{k-1} \binom{k-1}{i} (1 - q_k(p))^i q_k(p)^{k-1-i} \log_k(1+i) dp \\ &\quad + (1-t) \cdot \sum_{i=3}^k \binom{k-1}{i} (1-t)^i t^{k-1-i} \cdot \log_k EB(i). \end{aligned}$$

If all variables had completely distinct $y_{x,c'}$'s, then by Lemma 16 and Lemma 17 we would have that (\star) and in particular the running time of our algorithm is bounded by $O(k^{\text{cost}(k,t)n})$ for any choice of t . This would give us $O(2.22936^n)$ for $k=5, t=0.23$ and $O(2.64001^n)$ for $k=6, t=0.35$. We now deal with the case in which these are not distinct.

In the proof of Lemma 8 we faced the same problem and solved it by a simple application of Jensen's inequality to the concave function $\log_k(1+i)$. Unfortunately, the function $\log_k EB(i)$ is not concave (for $i=1, \dots, k$) due to its values on $i=1, 2$. Indeed, the left-hand side of the inequality in Lemma 17 is higher than the right-hand side if these variables are not distinct. On the other hand, when these variables are not distinct then the term of Lemma 16 is much smaller.

Consider the expression

$$\mathbb{E} \left[\sum_{i=1}^k Pr(x \in R_i) \cdot \log_k i + \sum_{i=3}^k Pr(x \in B_i) \cdot \log_k EB(i) \right] \quad (\star\star)$$

again. This time, we will assume that the variables $y_{x,c'}$ are not all distinct. Denote by $k' := |\{y_{x,c'} \mid c' \neq \varphi(x)\}| < k-1$ the number of such distinct variables, and by $j_1, \dots, j_{k'}$ their cardinalities (note that $\sum_{i=1}^{k'} j_i = k-1$). Consider the following expression.

$$\mathbb{E} \left[\int_0^t \sum_{b_1, \dots, b_{k'} \in \{0,1\}} q_k(p)^{k' - \sum_{i=1}^{k'} b_i} \cdot (1 - q_k(p))^{\sum_{i=1}^{k'} b_i} \cdot \log_k \left(1 + \sum_{i=1}^{k'} j_i b_i \right) dp \quad (\star\star\star) \right. \\ \left. + (1 - t) \cdot \sum_{b_1, \dots, b_{k'} \in \{0,1\}} t^{k' - \sum_{i=1}^{k'} b_i} \cdot (1 - t)^{\sum_{i=1}^{k'} b_i} \cdot \log_k EB \left(1 + \sum_{i=1}^{k'} j_i b_i \right) \right].$$

Expression $(\star\star\star)$ is a generalized form of $\text{cost}(k, t)$ and thus upper bounds $(\star\star)$ by the same arguments. Completely analysing the behaviour of Expression $(\star\star\star)$ for different partitions is rather technically involved and thus we simply enumerate over the few possible cases (for small values of k). In Section 4 we further discuss the possible improvements to the analysis of this section.

► **Theorem 18.** *We solve Unique $(6, 2)$ -CSP in $O(2.641^n)$ time.*

Proof. For the choice $t = 0.37$ we have that $\text{cost}(6, 0.35) = \log_6(2.64001)$, this choice of t minimizes $\text{cost}(6, t)$. We verify that for $t = 0.35$ Expression $(\star\star\star)$ is always lower than $\text{cost}(6, 0.35)$ and thus finish, as this implies that for every variable Expression $(\star\star)$ is bounded by $\text{cost}(6, 0.35)$.

- For the partition $(j_1, j_2, j_3, j_4) = (2, 1, 1, 1)$ the value of $(\star\star\star)$ in $t = 0.35$ is $\log_6(2.62023)$.
- For the partition $(j_1, j_2, j_3) = (2, 2, 1)$ the value of $(\star\star\star)$ in $t = 0.35$ is $\log_6(2.61171)$.
- For the partition $(j_1, j_2, j_3) = (3, 1, 1)$ the value of $(\star\star\star)$ in $t = 0.35$ is $\log_6(2.58391)$.
- For the partition $(j_1, j_2) = (3, 2)$ the value of $(\star\star\star)$ in $t = 0.35$ is $\log_6(2.60366)$.
- For the partition $(j_1, j_2) = (4, 1)$ the value of $(\star\star\star)$ in $t = 0.35$ is $\log_6(2.54819)$.
- For the partition $(j_1) = (5)$ the value of $(\star\star\star)$ in $t = 0.35$ is $\log_6(2.55566)$. ◀

► **Theorem 19.** *We solve Unique $(5, 2)$ -CSP in $O(2.232^n)$ time.*

Proof. For the choice $t = 0.23$ we have that $\text{cost}(5, 0.23) = \log_5(2.22936)$, this choice of t minimizes $\text{cost}(5, t)$. This time, unfortunately, for $t = 0.23$ Expression $(\star\star\star)$ is not always lower than $\text{cost}(5, 0.23)$. In particular, it is for every partition except of $(j_1) = 4$.

- For the partition $(j_1, j_2, j_3) = (2, 1, 1)$ the value of $(\star\star\star)$ in $t = 0.23$ is $\log_5(2.21658)$.
- For the partition $(j_1, j_2) = (2, 2)$ the value of $(\star\star\star)$ in $t = 0.23$ is $\log_5(2.21983)$.
- For the partition $(j_1, j_2) = (3, 1)$ the value of $(\star\star\star)$ in $t = 0.23$ is $\log_5(2.20499)$.
- For the partition $(j_1) = (4)$ the value of $(\star\star\star)$ in $t = 0.23$ is $\log_5(2.24925)$.

Denote by α the fraction of variables for which the $y_{x,c'}$ variables are all the same (i.e., variables with the only problematic partition). With the choice $t = 0.23$ the running time of our algorithm on a formula is $O\left((2.22936^{(1-\alpha)} \cdot 2.24925^\alpha)^n\right)$. On the other hand, if α is large we can simply run the regular PPSZ algorithm and gain much. We see that by setting $t = 1$ and computing Expression $(\star\star\star)$ for the same partition $(j_1) = (4)$, gives a value of $\log_5(2.01077)$. Thus, running regular PPSZ would give us a running time of $O\left((2.25303^{(1-\alpha)} \cdot 2.01077^\alpha)^n\right)$. We can therefore try both options ($t = 0.23$ or $t = 1$) simultaneously and thus get the running time of the faster one. Both expressions balance at $\alpha = 0.08612$, giving us a running time of $O(2.23107^n)$. ◀

Computation identical to this of Theorem 18 gives running time of $O(3.042^n)$ for $k = 7$ with $t = 0.44$.

4 Improvements and Limitations

In this section we sketch a possible improvement to the analysis of Section 3. The purpose of this section is not to tighten the upper bound but to explain the limitations of our algorithm and to convince that even with a tight analysis of Algorithm 2 it will achieve running times that are only slightly better than these we get in Section 3. In particular, if getting $O((2 - \varepsilon)^n)$ time for $(5, 2)$ -CSP is possible, new algorithmic tools are likely required.

Consider the expression $\prod_{i=1}^k i^{\mathbb{E}[R_i]} \cdot \prod_i EB(i)^{\mathbb{E}[B_i]}$ (\star) proven in Section 3 to upper bound the running time of our algorithm. In Lemma 16 we give a likely tight bound for the term involving $\mathbb{E}[R_i]$, yet in Lemma 17 we settle for a PPZ-type bound for $\mathbb{E}[B_i]$ in which we consider only the events in which the variables $y_{x,c'}$ themselves appear before time t and not the events in which they are decided by that time. The reason for this discrepancy becomes clear in the rest of the analysis. Due to the non-concave objective function in i , we can no longer assume independence between the events of each $y_{x,c'}$ being decided. Nevertheless, later in Theorem 18 and Theorem 19 we observe that while the term involving $\mathbb{E}[B_i]$ indeed gets worse with dependencies, the other term involving $\mathbb{E}[R_i]$ gets significantly better with them and thus can cover for those dependencies. Ideally, then, the simple PPZ-type bound of t in Lemma 17 can be replaced with the PPSZ-type bound of $q_k(t)$ in the total bound. This would result in the following tighter cost function.

$$\begin{aligned} \tilde{\text{cost}}(k, t) := & \int_0^t \sum_{i=0}^{k-1} \binom{k-1}{i} (1 - q_k(p))^i q_k(p)^{k-1-i} \log_k(1+i) dp \\ & + (1-t) \cdot \sum_{i=3}^k \binom{k-1}{i} (1 - q_k(t))^i q_k(t)^{k-1-i} \cdot \log_k EB(i). \end{aligned}$$

With this ideal cost function, we would get running times of $O(2.223^n)$ for Unique $(5, 2)$ -CSP (for $t = 0.32$) and $O(2.628^n)$ for Unique $(6, 2)$ -CSP (for $t = 0.46$).

5 Conclusions and Open Problems

In Section 3 we presented an algorithm for Unique $(k, 2)$ -CSP with a running time of $O(2.232^n)$ for $k = 5$. In Section 4 we argued that even with an ideal analysis, the bound we get for $k = 5$ is only the slightly better $O(2.223^n)$. Thus, it remains open and would likely require new algorithmic tools to show that $(5, 2)$ -CSP can be solved in $O((2 - \varepsilon)^n)$ time, or alternatively to rule out the existence of such algorithm by reductions to popular conjectures. More generally, we raise the following open problem.

► **Open Problem.** *What is the maximal k such that $(k, 2)$ -CSP can be solved in $O((2 - \varepsilon)^n)$ time?*

The main algorithmic observation in this paper is in fact a general insight regarding the behaviour of PPSZ-type algorithms. The Beigel-Eppstein algorithm [1] only works for $(k, 2)$ -CSP. On the other hand, the PPSZ-type algorithm [9] generalizes to $b > 2$ and is in fact currently the fastest algorithm for Unique (a, b) -CSP with $b > 2$ and any a . Using the tools we introduced in this paper, it should be possible to turn any faster algorithm for (a, b) -CSP for a specific (a, b) into a faster (a', b) -CSP algorithm for all $a' > a$.

Another follow-up question is whether our algorithm can be generalized to the non-unique $(k, 2)$ -CSP case.

References

- 1 Richard Beigel and David Eppstein. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, 2005.
- 2 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.
- 3 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *International Workshop on Parameterized and Exact Computation*, pages 75–85. Springer, 2009.
- 4 Tomás Feder and Rajeev Motwani. Worst-case time bounds for coloring and satisfiability problems. *Journal of Algorithms*, 45(2):192–201, 2002.
- 5 Fedor V Fomin and Petteri Kaski. Exact exponential algorithms. *Communications of the ACM*, 56(3):80–88, 2013.
- 6 F.V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2010.
- 7 Thomas Dueholm Hansen, Haim Kaplan, Or Zamir, and Uri Zwick. Faster k -SAT algorithms using biased-PPSZ. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 578–589, 2019.
- 8 Timon Hertli. 3-SAT faster and simpler - unique-SAT bounds for PPSZ hold in general. *SIAM J. Comput.*, 43(2):718–729, 2014. Announced at FOCS’11.
- 9 Timon Hertli, Isabelle Hurbain, Sebastian Millius, Robin A Moser, Dominik Scheder, and May Szedlák. The ppsz algorithm for constraint satisfaction problems on more than two colors. In *International Conference on Principles and Practice of Constraint Programming*, pages 421–437. Springer, 2016.
- 10 Eugene L Lawler. A note on the complexity of the chromatic number problem, 1976.
- 11 Liang Li, Xin Li, Tian Liu, and Ke Xu. From k -sat to k -csp: Two generalized algorithms. *arXiv preprint*, 2008. [arXiv:0801.3147](https://arxiv.org/abs/0801.3147).
- 12 Shibo Li and Dominik Scheder. Impatient ppsz—a faster algorithm for csp. *arXiv preprint*, 2021. [arXiv:2109.02795](https://arxiv.org/abs/2109.02795).
- 13 Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than $2n$ steps. *Discrete Applied Mathematics*, 10(3):287–295, 1985.
- 14 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k -SAT. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98*, pages 628–637, 1998. [doi:10.1109/SFCS.1998.743513](https://doi.org/10.1109/SFCS.1998.743513).
- 15 Ramamohan Paturi, Pavel Pudlák, Michael E Saks, and Francis Zane. An improved exponential-time algorithm for k -SAT. *Journal of the ACM (JACM)*, 52(3):337–364, 2005.
- 16 Dominik Scheder. Ppz for more than two truth values—an algorithm for constraint satisfaction problems. *arXiv preprint*, 2010. [arXiv:1010.5717](https://arxiv.org/abs/1010.5717).
- 17 T Schoning. A probabilistic algorithm for k -SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 410–414. IEEE, 1999.
- 18 Patrick Traxler. The time complexity of constraint satisfaction. In *International Workshop on Parameterized and Exact Computation*, pages 190–201. Springer, 2008.
- 19 Gerhard J Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial optimization – eureka, you shrink!*, pages 185–207. Springer, 2003.
- 20 Or Zamir. Breaking the 2^n barrier for 5-coloring and 6-coloring, 2020.

